Escuela
Politécnica
Superior

App móvil para la gestión de astilleros

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Jorge Poveda Pérez

Tutor/es:
Estela Saquete Boro

Universitat d'Alacant
Universidad de Alicante

Julio 2020

TABLA DE CONTENIDO

INTRODUCCIÓN

El turismo náutico o chárter náutico [1] es una manera cada vez más popular de combinar el turismo de navegación con las vacaciones.

No sólo es una forma agradable de ver lugares únicos en el mundo, también es una industria muy rentable. Muchos turistas que disfrutan de la navegación combinan el turismo acuático [2] con otras actividades. El suministro de equipos y accesorios para estas actividades ha dado lugar a empresas para estos fines.

Estas empresas, que poseen una flota de embarcaciones y empleados afiliados al sector, ofrecen su alquiler a turistas interesados mediante un sistema de reservas. Tanto de las propias embarcaciones como de los patrones y capitanes o el material necesario para realizar ciertas actividades deportivas o de ocio.

Al introducir una aplicación tecnológica en este proceso, encontramos un nicho de mercado al que podemos tener acceso creando una herramienta capaz de mediar no solo entre empresas, también entre usuarios particulares que ofrezcan sus embarcaciones o servicios en nuestra aplicación, para que el resto de los usuarios y turistas puedan reservar directamente.

La aplicación se ha llamado EmbarcApp, y podemos encontrar su código fuente y descargarla desde el repositorio donde se ha desarrollado [3].

Resumen

El proyecto a desarrollar pretende operar en este sector, ofertando, a través de una aplicación móvil, viajes y travesías en distintos vehículos navegables, así como el alquiler de los mismos.

Los usuarios, a su vez, serán capaces de buscar, filtrar, gestionar y reservar estos viajes mediante una interfaz sencilla, intuitiva y moderna que garantice los patrones de User Experience (UX) [4].

Se ha implementado para las plataformas de Android e iOS, las cuales superan el 99% del mercado actual en telefonía [5] así como para navegadores web por medio de una PWA o Aplicación Web Progresiva [6].

Motivación

La propuesta del proyecto surge a raíz de la proposición de un particular, por lo que existe un interés externo en el proyecto que podría formalizarse mediante un contrato.

Esta situación ha ayudado a mejorar el producto final, que ha obtenido valor añadido gracias al feedback y la comunicación con las personas interesadas.

Estudio de mercado

Se han construido puertos desarrollados especialmente para los turistas náuticos en Europa, América del Sur y Australia. En nuestro país, concretamente, la mayor parte de las empresas relacionadas se encuentran en las zonas costeras del mediterráneo.

[7]

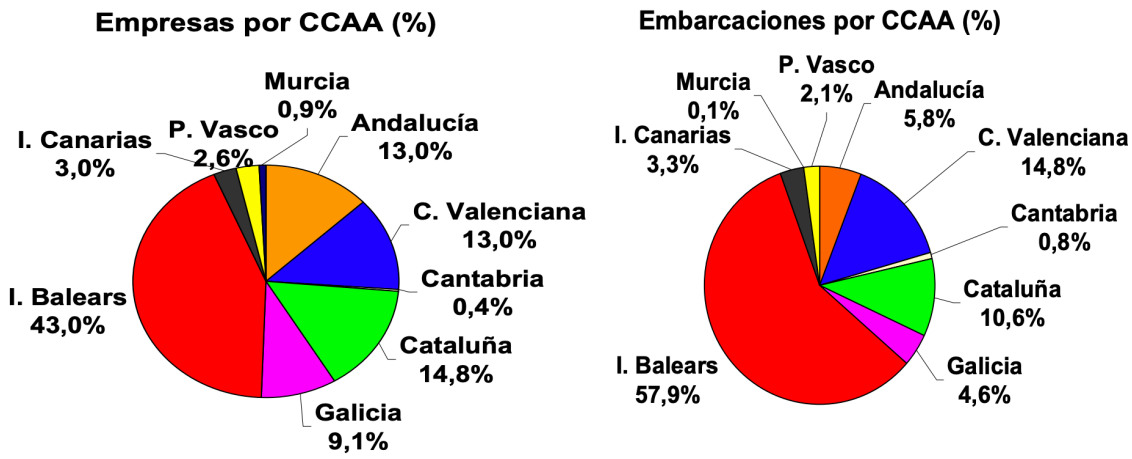

Ilustración 1: porcentaje de empresas dedicado al chárter náutico según Comunidad autónoma en referencia al porcentaje de número de embarcaciones dedicadas al sector por CCAA

Como ejemplo, el sector en Baleares, la CCAA con más volumen en el mercado de España, que posee 99 empresas dedicadas a ofrecer estos servicios y más de 1400 embarcaciones [7], ha generado 561 millones de euros y mas de 2500 puestos de trabajo de forma directa. [8]

Durante el estudio de mercado no se han encontrado aplicaciones móviles que ofrezcan un producto similar. Sí que existen páginas webs, pero teniendo en cuenta que cada vez es más evidente la importancia y el impulso que puede dar una App a un negocio, se cree que la accesibilidad que ofrecería el fruto de este proyecto puede tener ventaja sobre el resto de competidores. A continuación, veremos con más detalle algunos ejemplos de webs en este sector.

Nautal

Esta web [9] es representativa del tipo de páginas más común del sector. Se trata de una herramienta con unas funcionalidades similares a las que se pretenden implementar en nuestro caso. No obstante, las embarcaciones de recreo de las que dispone están asociadas a la empresa y nunca a solo particulares.

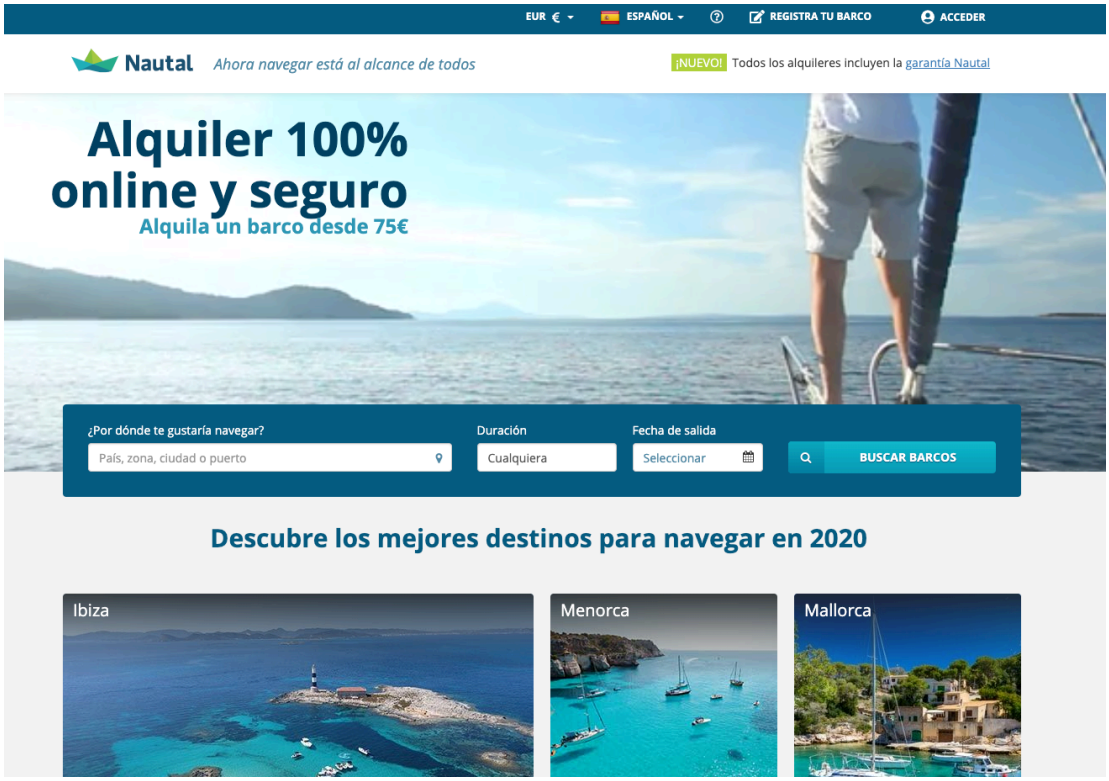

Ilustración 2: Portada web de Nautal

Nos ofrece un buscador que permite filtrar según distintas categorías, y ordenar por relevancia y/o precio.

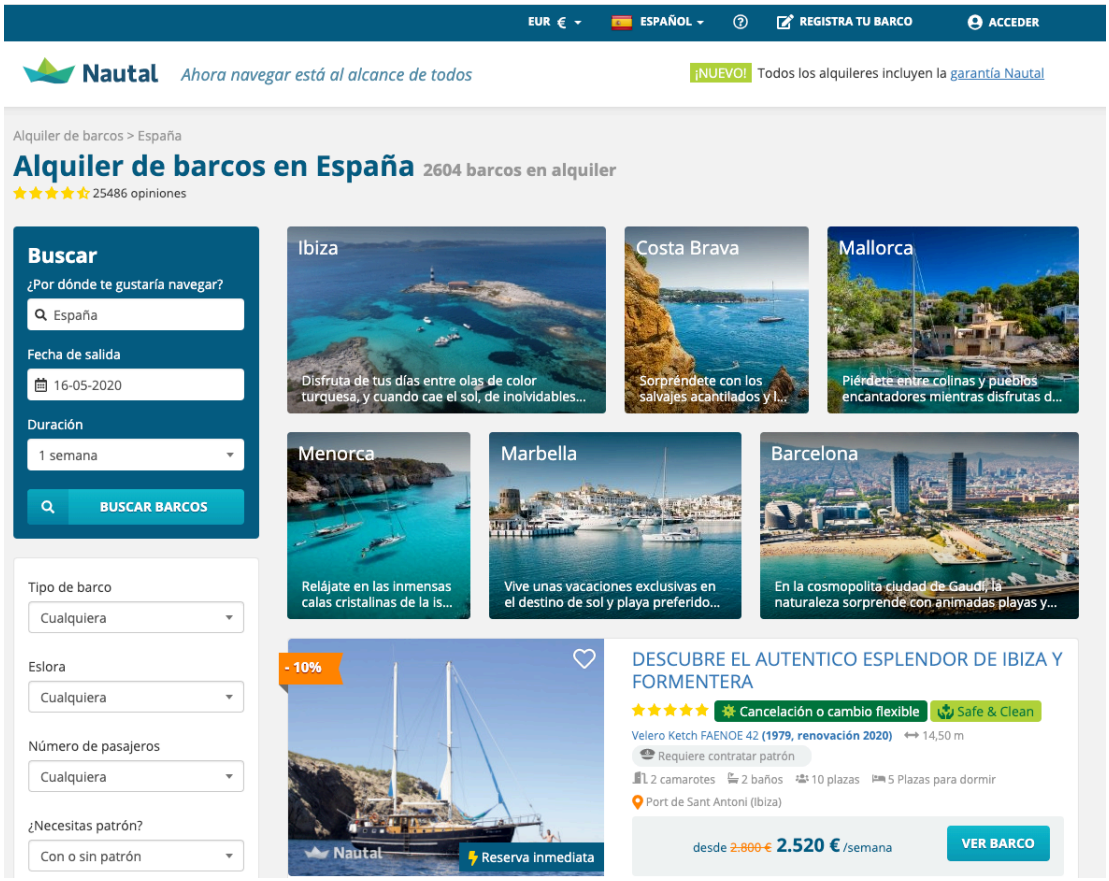

Ilustración 3: Búsqueda de ofertas en embarcaciones de la web Nautal

El proceso de reserva y gestión de la misma es muy similar al que pretendemos obtener en nuestra aplicación, donde un usuario registrado, consulta los detalles, añade a favoritos o reserva directamente un servicio para una fecha determinada.

Además, disponen de App móvil para iOS [10] y Android [11]. Estas aplicaciones poseen pocas descargas y malas críticas debido a su funcionamiento.

Al estar centrada solo en la gestión de las reservas por parte de los usuarios que proponen las ofertas, creemos que están evitando hacer verdadero uso de la accesibilidad que proporciona a los clientes finales una aplicación móvil.

y4ybooking

Esta página [12] solo opera en el extranjero, y al igual que Nautal, se ha seleccionado para analizar al pertenecer a un grupo de webs con una cierta característica.

Esta peculiaridad es que no cuenta con una base de datos propia, si no que las ofertas que muestra la página son consumidas junto a otros datos relevantes de una API externa (en este caso, ofrecida por Nausys [13]).

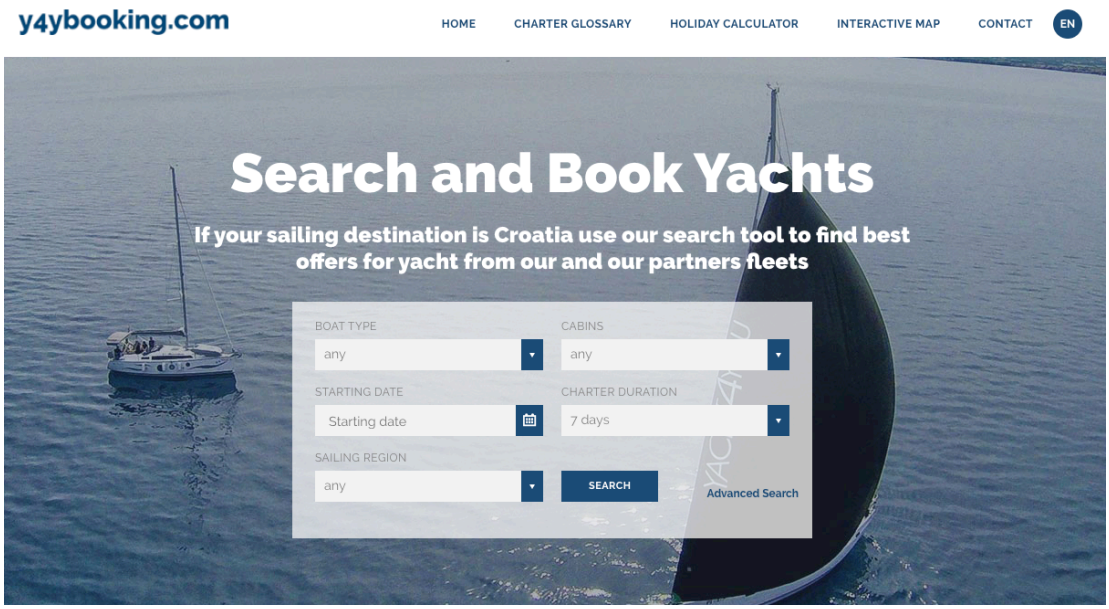

Ilustración 4: Portada de la web y4ybooking.com

Esta herramienta queda entonces relegada a un rol de mediador entre los verdaderos propietarios de las embarcaciones (que han ofertado sus servicios en Nausys) y los clientes finales.

La aplicación desarrollada busca incurrir en un papel de intermediario similar al que se acaba de explicar, por lo que se cree que la abundancia de este tipo de páginas solo hace que demostrar la viabilidad del proyecto.

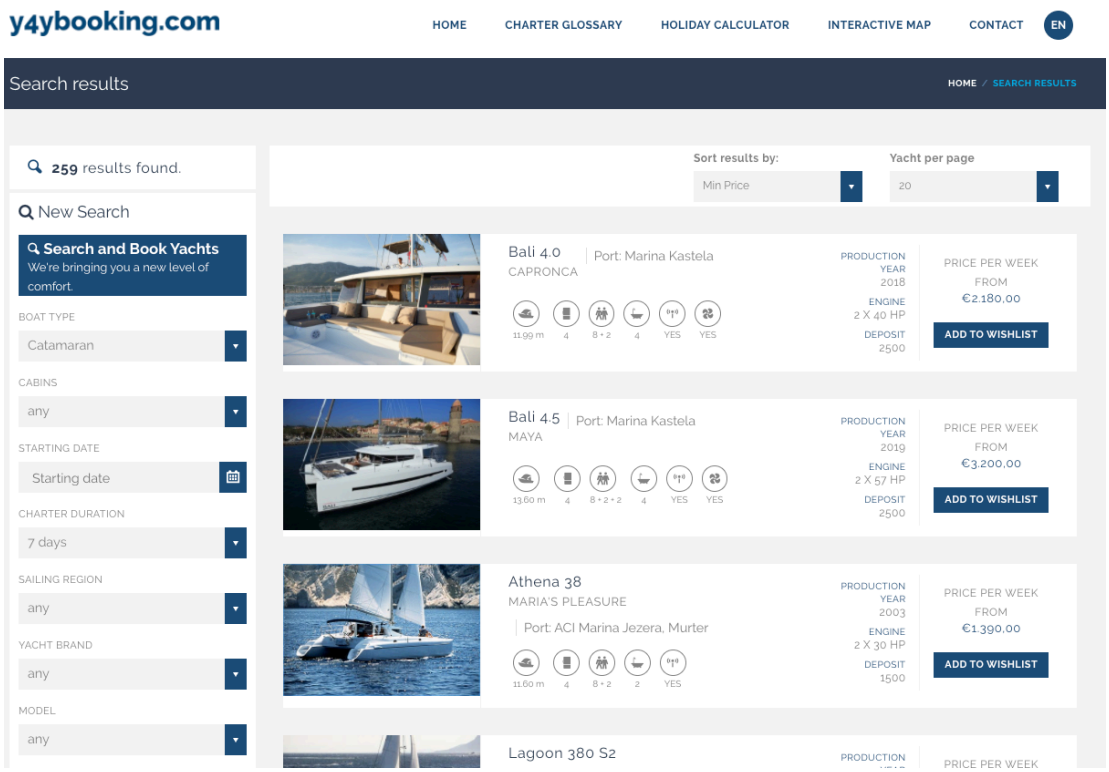

Ilustración 5: Búsqueda de ofertas en la web y4ybooking.com

Entre otras funcionalidades con, en principio, algo menos de relevancia, encontramos también un mapa interactivo que señala los puertos de los que contiene registros:

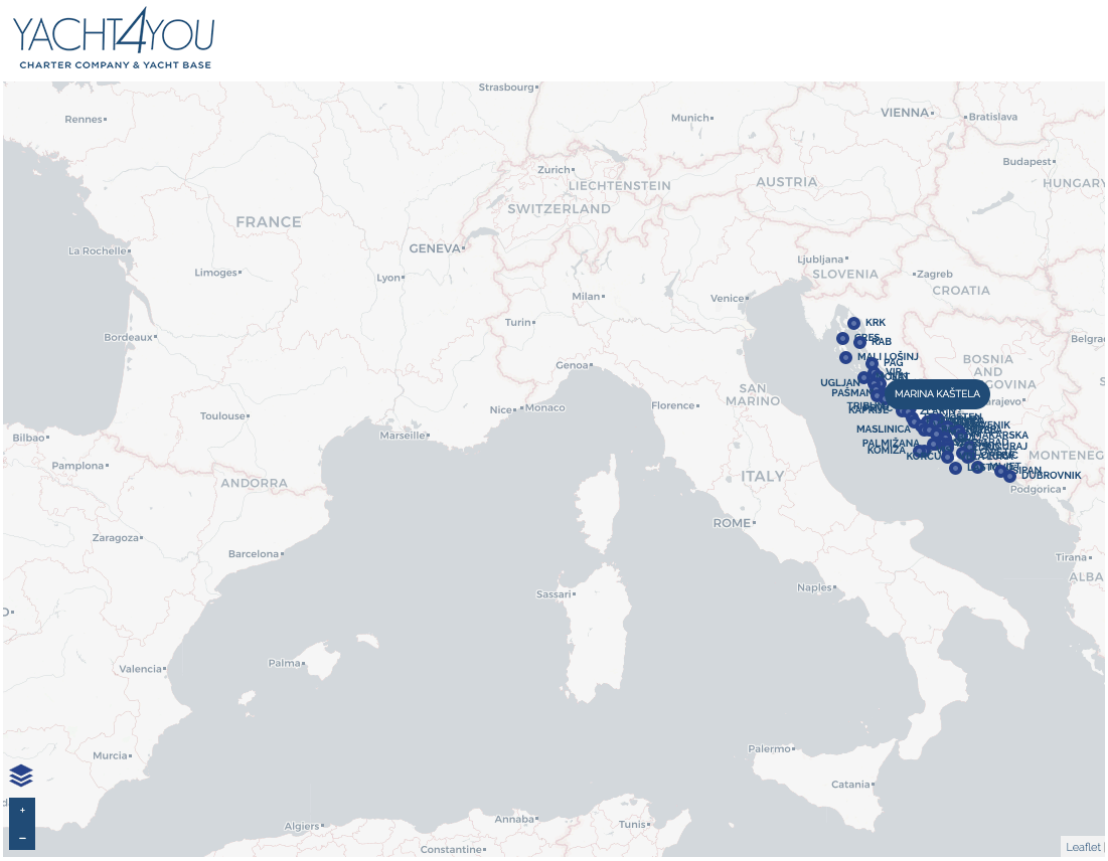

Ilustración 6: Mapa interactivo con las ofertas disponibles en y4ybooking.com

Esta herramienta es implementada por una librería JavaScript llamada LeafletJS [14] y usa el mapa con licencia abierta OpenStreetMap [15].

Hay un rasgo muy común en este grupo de páginas, y es que al tener que pagar una cuota a terceros por los datos de embarcaciones, suelen centrar todas sus ofertas en una misma zona relativamente pequeña.

En resumen, es interesante resaltar la posibilidad de obtener una porción del mercado tan solo ofreciendo un producto software visualmente atractivo que simplemente se encargue de interceder entre los verdaderos propietarios y los clientes finales.

LogiTravel

LogiTravel [16] se trata de un portal con bastante popularidad de turismo y viajes.

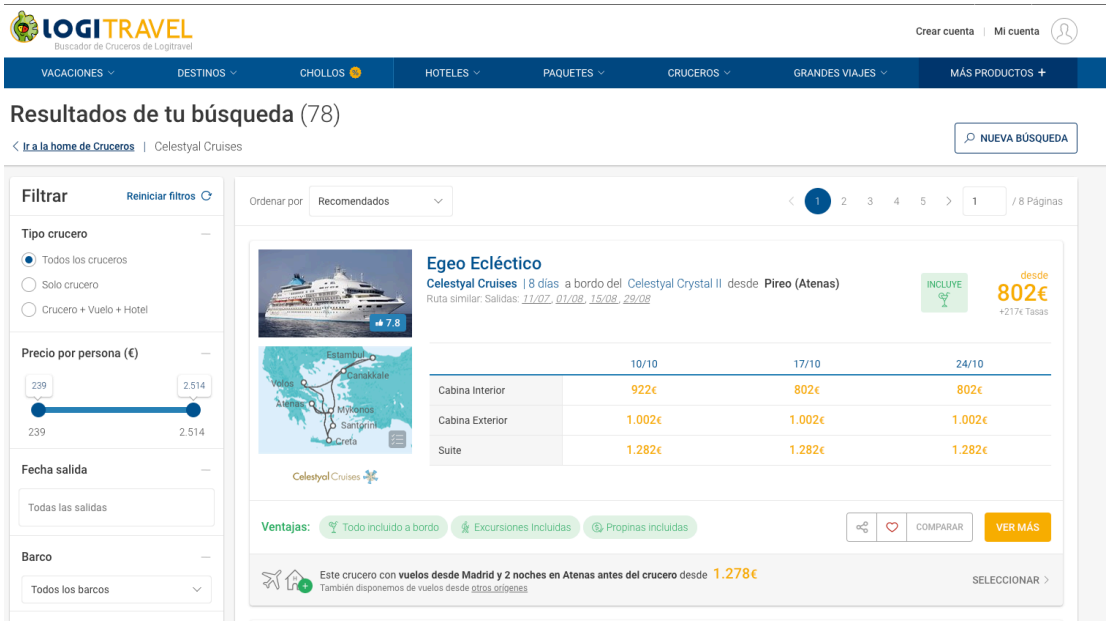

Ilustración 7: Búsqueda de ofertas en embarcaciones marítimas en Logitravel

Esta web nos muestra distintos precios para la misma nave, donde podemos observar que hay diferentes tarifas que varían según la estancia o la calidad del servicio para una misma embarcación.

También permite acumular otros servicios mediante ofertas como hoteles o viajes en avión.

En cuanto a turismo por mar solo poseen servicios en cruceros, por lo que, de las webs analizadas, es la más alejada a nuestro propósito.

Aun así, creo que es una página relevante para el análisis, ya que nos permite ver de forma palpable el éxito que pueden conllevar los negocios de turismo marino.

HERRAMIENTAS, TECNOLOGÍAS Y LENGUAJES DE PROGRAMACIÓN EMPLEADOS

Lenguajes de programación y Frameworks

Teniendo en cuenta que se trata de un desarrollo llevado a cabo por una sola persona y uno de sus requerimientos es la necesidad de disponer de versiones para los distintos SOs, no es práctico implementar la aplicación de móvil en el lenguaje nativo. Es interesante anotar que Android trabaja con Java o Kotlin, mientras que iOS lo hace con Objective C o Swift.

Por estas razones se ha decidido desarrollar de forma híbrida haciendo uso de tecnologías como Cordova e Ionic que se encargan de transformar el código a un lenguaje que ambas plataformas puedan inyectar en sus WebViews.

La clave del desarrollo híbrido (web, HTML5) se encuentra en estos WebViews que nos permiten navegar de forma segura por una web desde dentro de una aplicación de móvil.

Por si no fuera suficiente razón, el hecho de utilizar tecnologías web, nos permite también generar una versión PWA de la aplicación, por lo que, con un solo desarrollo, estaríamos cubriendo todas las plataformas requeridas.

Apache Cordova [17]

Se trata del framework que hace posible que un desarrollo web estándar funcione como cross-platform, permitiendo, implementar un desarrollo que finalmente será compatible con los distintos sistemas operativos de los principales teléfonos móviles.

Esta tecnología se encarga de encapsular el proyecto web dependiendo de la plataforma a la que va dirigida. En caso de querer modificar este comportamiento, siempre podemos hacer uso del archivo config.xml, que nos permite customizar comportamientos para nuestra aplicación como el motor web que la ejecuta, protocolos de red que acepta, versión mínima requerida del SO para que el dispositivo la ejecute,…

También hace uso de las APIs nativas de los SO para acceder a funciones básicas o de hardware como la cámara, el altavoz, micrófono, geolocalización, …

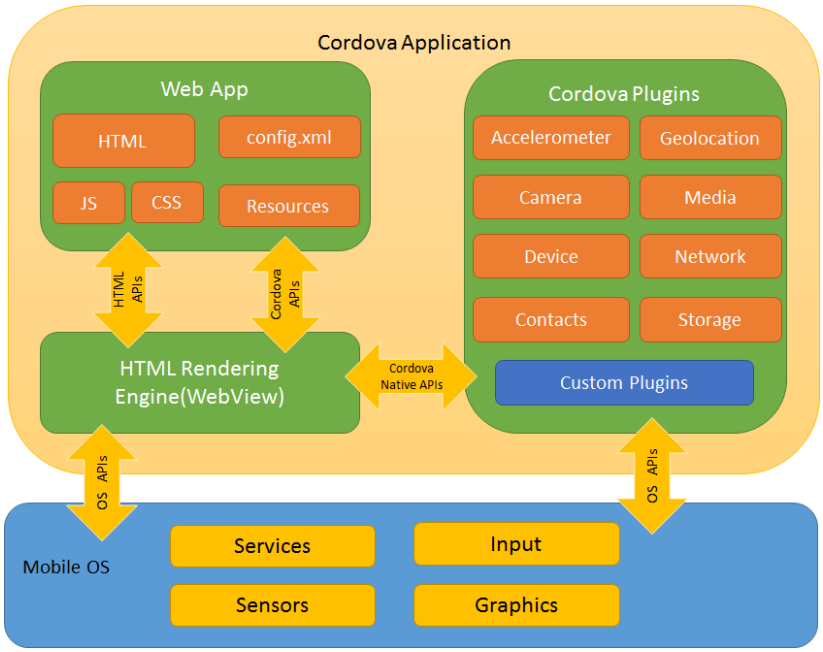

Ilustración 8: Gráfico explicativo de Apache Cordova

En nuestras aplicaciones podemos hacer uso de los plugins oficiales de cordova (Core Plugins) o de los distintos que provee la comunidad, así como implementar los nuestros propios. Todo ello con el objetivo de acceder a estas funciones nativas y traducir su respuesta a la WebView que finalmente, se renderiza en nuestros dispositivos.

Ionic [18]

La aplicación ha sido desarrollada usando Ionic.

Ionic es un kit de desarrollo software (SDK) de código libre. Se trata de un framework basado en tecnologías web como JavaScript, HTML y CSS.

Haciendo uso de Apache Cordova o Capacitor es capaz de transportar una implementación web a distintas plataformas:

- PWA: Progressive Web App.
- Aplicaciones de escritorio.
- Android, iOS, Windows Phone.

En cuanto a este último apartado, conforma una opción muy popular para el desarrollo móvil híbrido, por lo que dispone de mucha documentación e información tanto oficial, como de desarrolladores particulares.

Su escalabilidad le permite, a su vez, hacer uso de distintas plataformas de desarrollo web como Vue, Angular, React,…

Ilustración 9: Gráfico explicativo de Ionic Framework

Para nuestro caso, se ha utilizado Angular debido a su potencia, estabilidad y amplia documentación.

La curva de aprendizaje de este lenguaje ha permitido que el desarrollo avance de forma más acelerada tras la codificación de los primeros requisitos. De modo que, tras un tiempo muy breve, la facilidad para programar en este lenguaje ha ascendido de forma considerable.

Angular [19]

Es una popular plataforma de desarrollo y de diseño de aplicaciones web basada en HTML, CSS y a pesar de que originalmente hacia uso de JavaScript, a partir de Angular2, funciona con TypeScript.

Con el tiempo, este cambio ha demostrado no tratarse de un simple hecho aislado ya que Angular ha gozado de continuas actualizaciones que han mejorado considerablemente su rendimiento y fundamentos.

Dichos fundamentos se constituyen en:

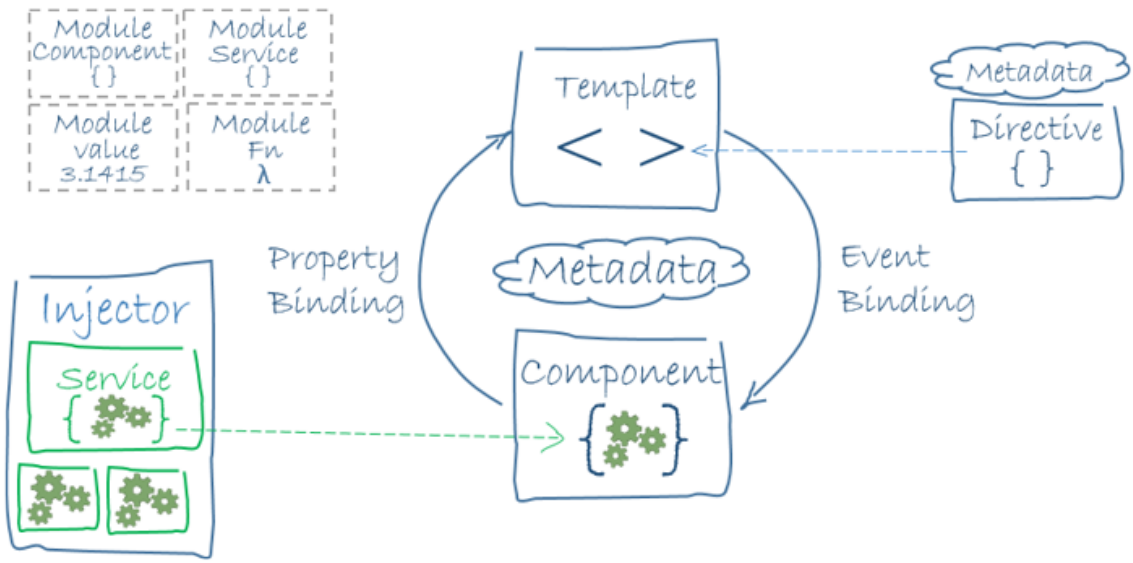

Ilustración 10: Gráfico explicativo de Angular

Módulos

Parecidos a las clases convencionales que encontramos en cualquier otro lenguaje. Estos módulos (NgModule) aportan el contexto de compilación para los *Componentes*.

Los Módulos agrupan todo el código relacionado en distintos conjuntos funcionales, estos conjuntos suponen la *Aplicación*.

Toda *Aplicación* posee un *Módulo Raíz* en el que ésta se arranca por primera vez, precargando los elementos necesarios e instanciando la navegación.

Componentes

Los *Componentes* definen las *Vistas*, formadas por elementos que pueden ser escogidos por Angular para ser modificados y alterados en concordancia con la lógica programada y/o datos de los que dispone.

Los *Componentes* consumen servicios (*Providers*), que encapsulan una funcionalidad especifica que no está relacionada directamente con las propias Vistas.

Los metadatos de un Componente lo asocian con una plantilla que define una Vista.

Providers

Los *Providers* pueden inyectarse dentro de los *Componentes* como dependencias, haciendo el código más escalable, modularizado y eficiente.

Los metadatos en un Provider aportan información que Angular necesita para hacer que los Componentes estén disponibles a través de la inyección de dependencias.

Decorators

Módulos, *Componentes* y *Providers* son clases que utilizan *Decorators*.

Los *Decorators* definen su tipo y aportan metadatos que dicen a Angular cómo usarlos.

Una plantilla combina HTML plano con Directivas de Angular y referencias de enlace (binding) que le permite a Angular modificar el HTML antes de que se renderice.

Navigator/Router

De forma usual, los componentes de una aplicación definen muchas vistas, que quedan ordenadas de forma jerárquica.

Angular provee el servicio de enrutado para ayudar a definir la navegación entre las distintas vistas.

Herramientas

Visual Studio Code

Se ha hecho uso de Visual Studio Code para implementar el proyecto en Ionic (desarrollo web) que luego eran transportadas a proyectos compatibles con las distintas plataformas.

Code posee una interfaz muy simple en la que desarrollar nuestro código y muchas funciones que lo facilitan.

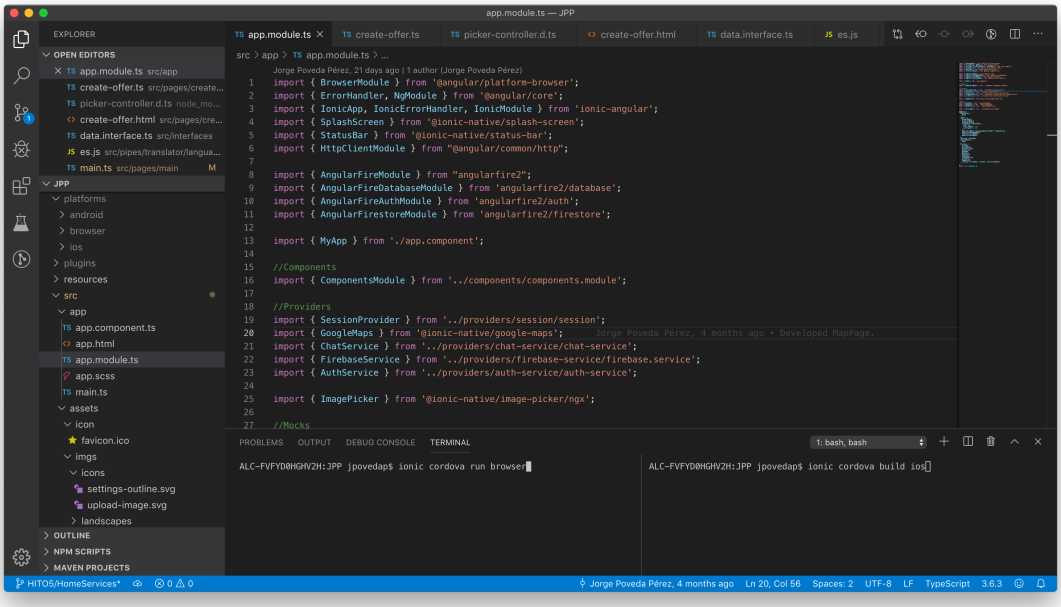

Ilustración 11: Captura de un proyecto Ionic abierto en Visual Studio Code

El terminal en línea resulta muy cómodo, así como las distintas extensiones que pueden añadirse a nuestra herramienta, entre las que destacan las más útiles como Gitlens:

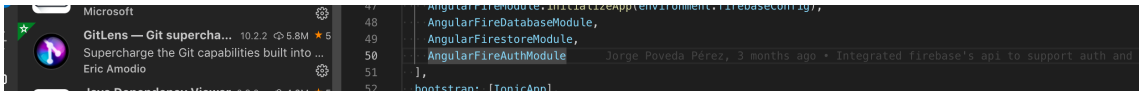

Ilustración 12: Captura de Visual Studio Code, donde se aprecia la extensión GitLens y un ejemplo de su funcionamiento

Que nos muestra “inline” el commit en el que se realizó el último cambio sobre la línea en la que tenemos el puntero y la fecha en la que tuvo lugar.

También encontramos otras extensiones que nos permiten formatear automáticamente el código para que siga los estándares y favorezca la limpieza y legibilidad del código independientemente del lenguaje en el que se programe.

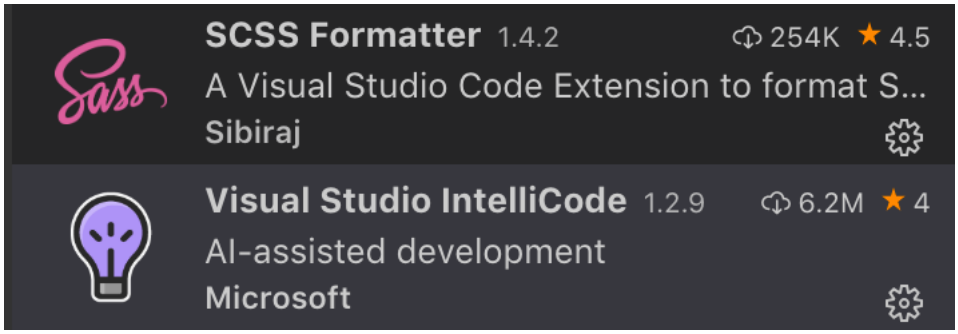

Ilustración 13: Captura de las extensiones de Visual Studio Code; SCSS Formatter e IntelliCode

O un asistente IA que nos ayude a autocompletar las palabras con las opciones más probables y/o disponibles.

Android Studio

El proyecto en su versión para Android es generado y ejecutado por Android Studio. Para crearlo, lanzaríamos por consola el comando:

ionic cordova build android

Los CLI (Command Line Interface) [20] de Ionic y Cordova nos permitirían entonces disponer de un proyecto compatible con Android Studio de forma automática. Además, podríamos acelerar más el proceso de creación de una versión final a través de la configuración del config.xml que tenemos en la raíz de nuestro proyecto Ionic.

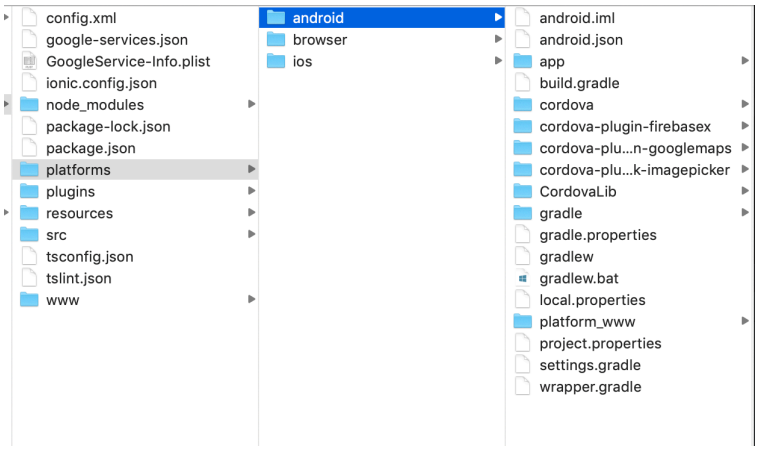

Ilustración 14: Directorio generado por Cordova que contiene un proyecto compatible con Android Studio

Android Studio es quien se encarga de descargar y encapsular las librerías necesarias para que nuestra aplicación funcione en dispositivos Android.

Este proceso es llevado a cabo por Gradle [21] un sistema de automatización de construcción de proyecto que va integrado por defecto en el propio Android Studio.

Android Studio a través de su módulo Android Virtual Device Manager también nos permite emular dispositivos en los que probar las aplicaciones. Esta funcionalidad es especialmente útil para debugear o testear funcionalidades cuyo comportamiento depende del SO o sus cualidades nativas (cámara, geolocalización emulada,) y hacerlo sin tener que depender de un dispositivo físico.

XCode

A su vez, la plataforma iOS hace uso de XCode para la generación de una versión final en iOS. El sistema de gestión y descarga de paquetes que utiliza es CocoaPods.

Gracias al comando:

ionic cordova build ios

Cordova podría generarnos un proyecto (.xcworkspace) compatible con XCode en la carpeta platforms/ios de nuestro directorio.

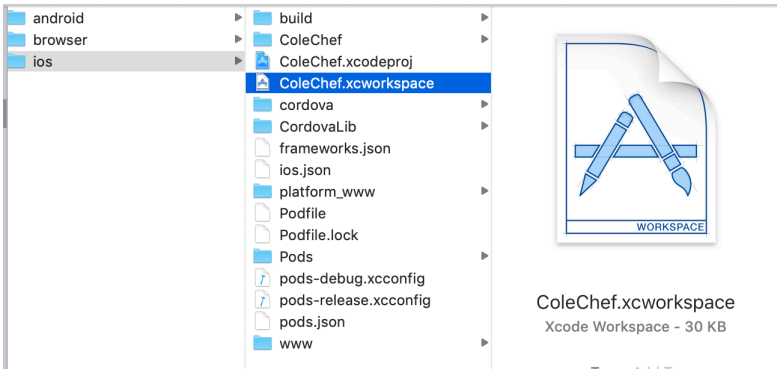

Ilustración 15: Directorio generado por Cordova que contiene un proyecto compatible con XCode

Una vez hecho, podríamos lanzar la aplicación para probarla en sus simuladores.

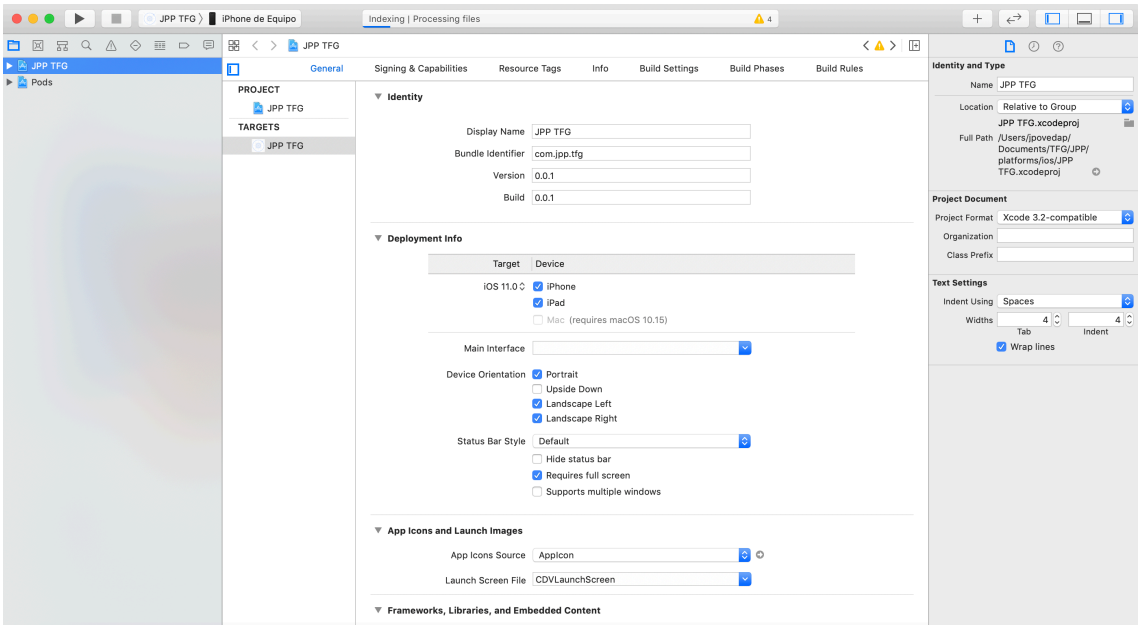

Ilustración 16: Captura del proyecto tras abrirlo con XCode

Para añadir a nuestra aplicación ciertas funcionalidades debemos darlas de alta en el espacio de Signing & Capabilities. Este es el caso de las notificaciones push por ejemplo que hemos implementado en nuestro proyecto:

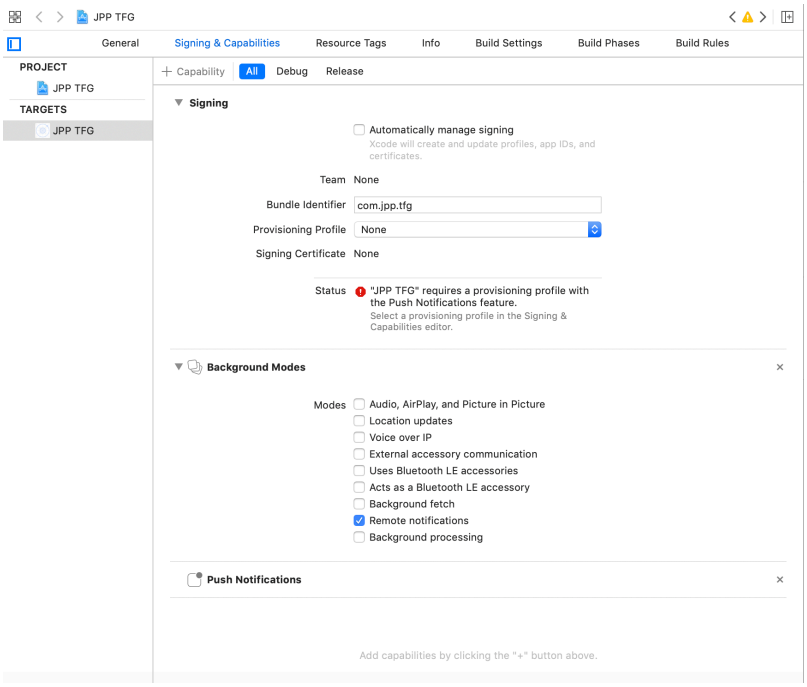

Ilustración 17: Captura del apartado de configuración de proyecto Signing & Capabilities de XCode

En caso de querer crear una versión necesitaríamos configurar el proyecto añadiéndole una entidad validada como firmante.

Esta entidad, tiene que crear un certificado desde la consola de desarrollador de Apple, y tras descargarlo, adjuntarlo en la configuración del proyecto.

Sourcetree

Es una aplicación de escritorio que nos permite manejar nuestros repositorios de forma mucho más intuitiva.

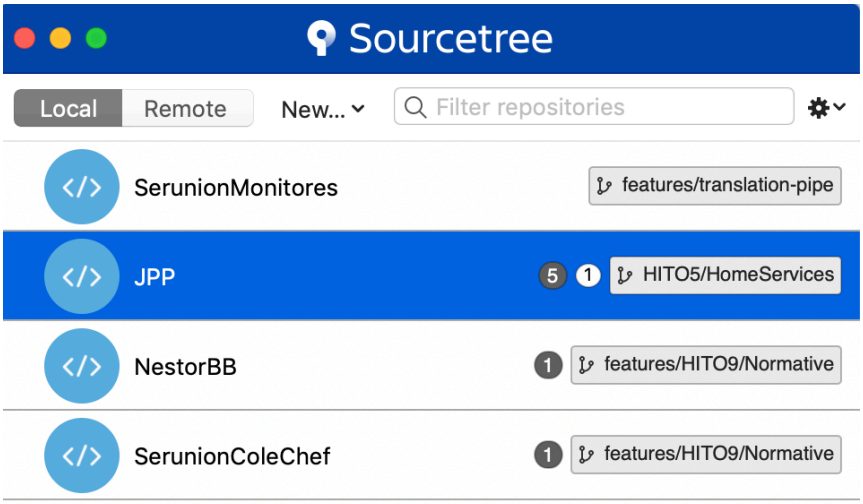

Ilustración 18: Captura del programa Sourcetree mostrando un listado de los repositorios asociados

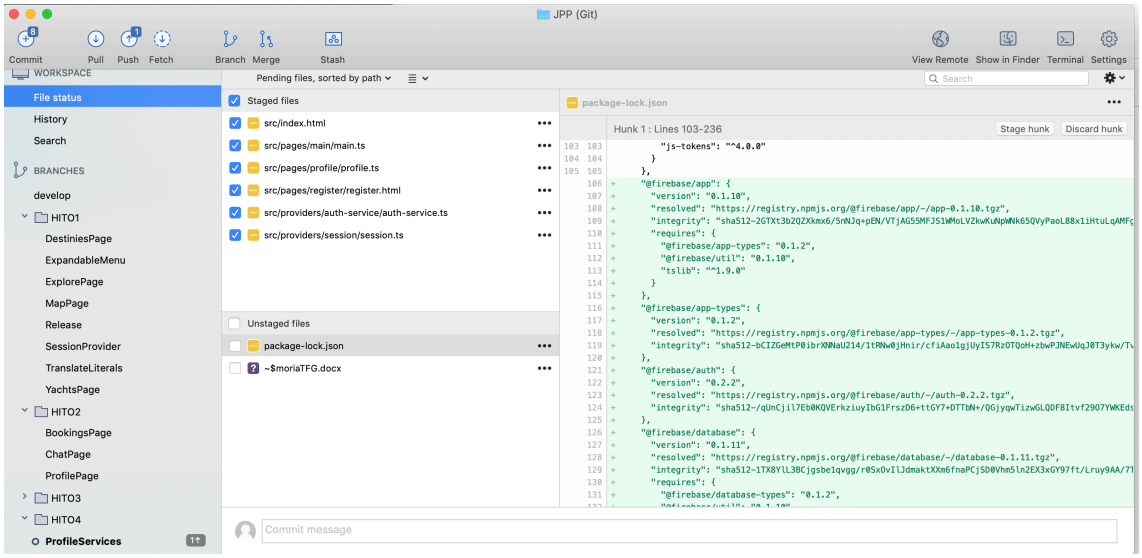

Ilustración 19: Captura de un repositorio abierto con Sourcetree

Con Sourcetree, dejamos de utilizar comandos de terminal para las acciones de Git, que son intercambiadas por botones y acciones en una interfaz visual.

Gracias a esta aplicación tenemos mucha más facilidad y seguridad al realizar cualquier acción en la que intervenga el control de versiones. El uso frecuente y la experiencia consiguen que los posibles errores que puedan surgir por el factor humano se minimicen notablemente.

En cuanto al desarrollo del proyecto, su uso ha sido esencial para acelerar y agilizar la implementación de las nuevas funcionalidades y mejoras.

Node Package Manager (NPM) [22]

Como su nombre indica, se trata de un sistema de paquetería implementado en sus inicios para Node.js.

Es un CLI que nos permite descargar los paquetes externos que queremos importar en nuestro proyecto. Estas dependencias serían instaladas en nuestra aplicación en una carpeta llamada node_modules.

Todos los paquetes NPM poseen un package.json, donde al menos se encuentran dos atributos, name y version.

Todo proyecto Ionic dispone de un package.json en su directorio raíz que incluye los identificadores y versiones de paquetes requeridos para hacerlo funcionar.

Usando el comando:

$ npm install

Descargaríamos todas estas librerías en un directorio de nuestro proyecto llamado node_modules.

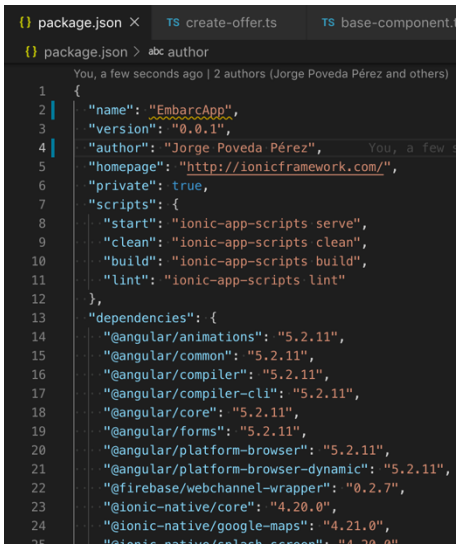

Ilustración 20: Ejemplo de un archivo package.json

Google Chrome

Durante el desarrollo, podemos lanzar la aplicación en el navegador (browser) para observar los cambios realizados y probar las funcionalidades antes de generar una versión.

Para ello, tendríamos que lanza el comando:

$ ionic cordova run browser

Esta orden, generaría un contenedor en nuestro ordenador en uno de sus puertos (8000 por defecto, si está ocupado, se incrementa en sucesión) con la aplicación. Dicho comando tarda entre 15-30s en finalizar por lo que resulta muy rápido durante la fase de desarrollo el probar los cambios en nuestra aplicación.

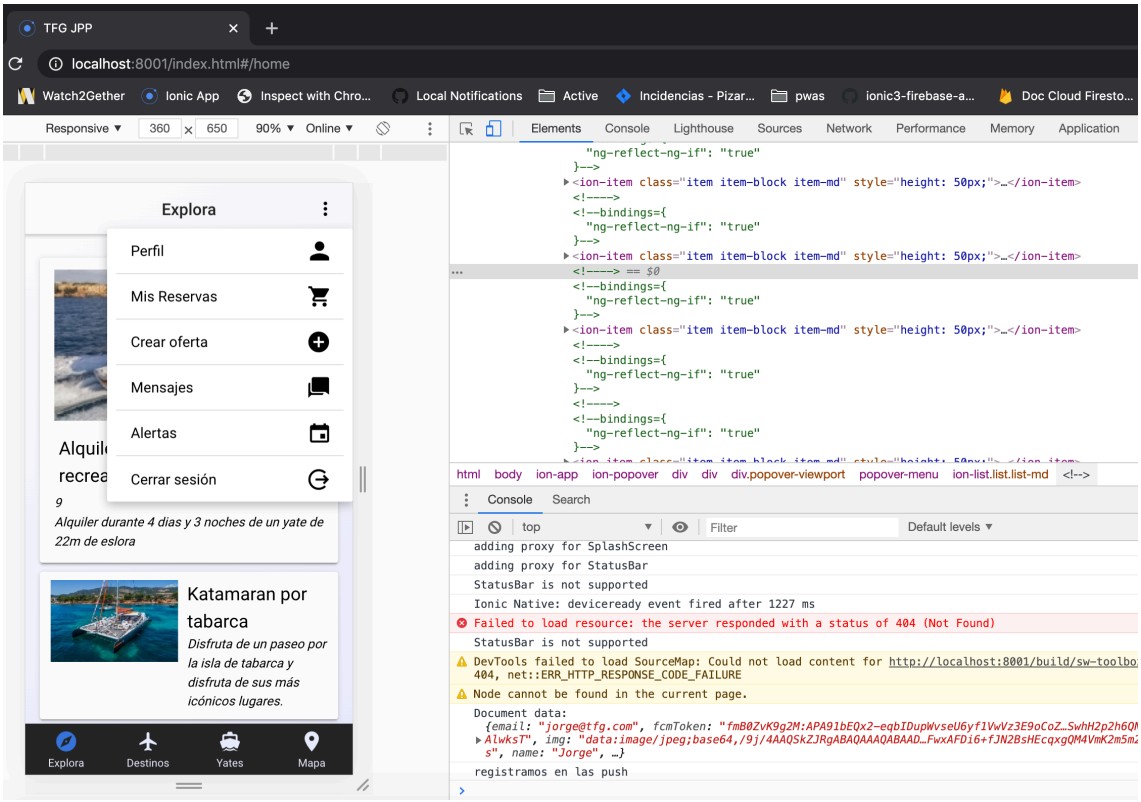

Ilustración 21: Captura de Google Chrome tras abrir el Inspector de elementos

Entre las funciones mas importantes que ofrece Chrome encontramos:

- Inspector HTML y de estilos CSS que permite modificar en caliente los estilos. Este método acelera considerablemente el desarrollo, diseño y ajuste de los estilos y la vista.
- Consola en línea donde observar los errores o cadenas de control que hayamos insertado durante el desarrollo para poder visualizar el estado de la aplicación.
- Herramienta de debug con la que añadir breakpoints y facilitar la resolución de incidencias.

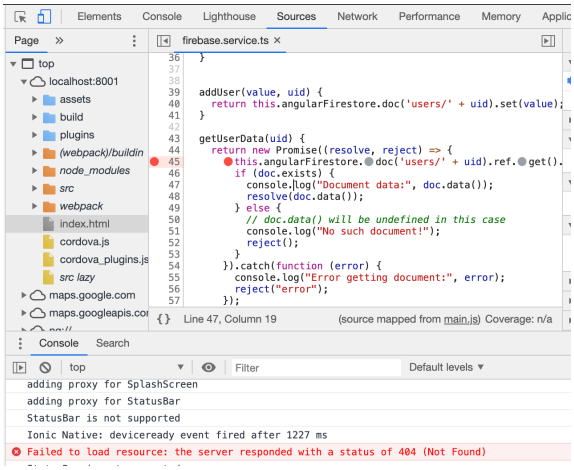

Ilustración 22: Captura del módulo Sources de Chrome

- Módulo Network que se dedica a observar la red. Captura y muestra los datos de cada petición generada. Entre los más relevantes encontraríamos el estado de las peticiones HTTP (200 OK, 404 Not Found,..), el tiempo que ha tardado en completarse, el Endpoint (URL) al que se realiza, el cuerpo del mensaje (si lo tuviese),...

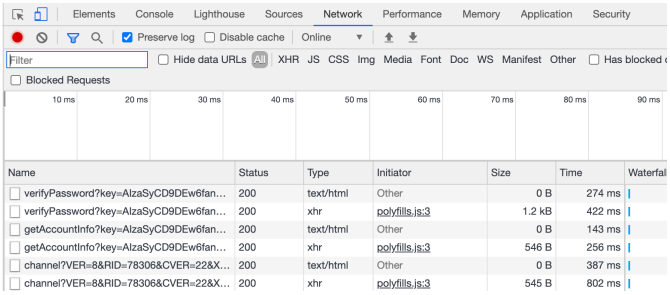

Ilustración 23: Captura del apartado Network de Chrome

- Gestor de distintos tamaños y resoluciones en dispositivos simulados con los que visualizar el comportamiento de nuestra aplicación según las distintas ratios de pixels.

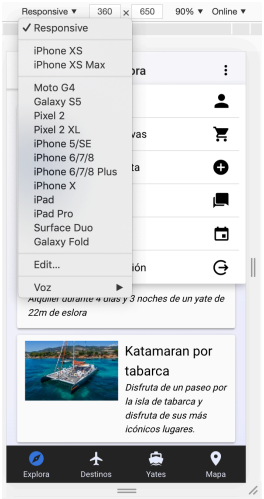

Ilustración 24: Gestor de tamaño de dispositivos de Chrome

Hay que destacar que Google Chrome tiene su contraparte en Safari o su última versión “Safari Technology Preview” para el caso de estar debugando un simulador de iOS y posee prácticamente las mismas funcionalidades.

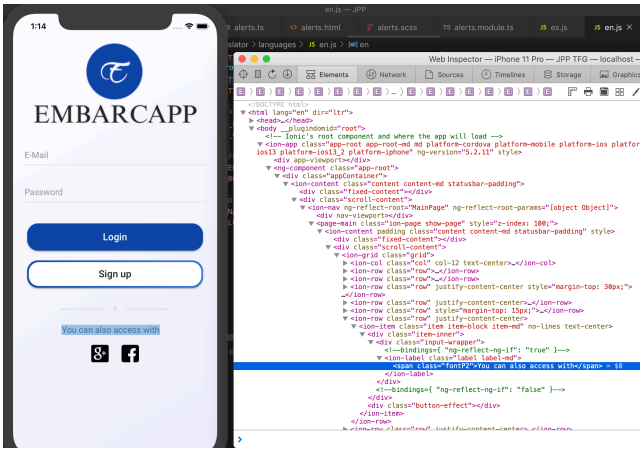

Ilustración 25: Captura de un simulador iOS con el inspector de Safari a un lado

PLANIFICACIÓN DEL PROYECTO

El código fuente del proyecto está contenido en el gestor de repositorios BitBucket [23]. Se trata de un sistema de control de versiones basado en Git y que, además, integra otras muchas funciones útiles que nos permiten conocer el estado y gestionar nuestro proyecto en todo momento.

Control de versiones (Git) [24]

El desarrollo ha sido planificado por Hitos que añaden una serie de “features” o lista de requisitos. Se trata de un sistema de “release” (entrega) incremental.

Para llevar a cabo este modelo de trabajo se ha generado una rama master, y otra develop. La rama master contiene el proyecto en su versión “estable”. La rama develop se trata de la rama a partir de la cuál se generan las versiones durante el desarrollo para comenzar el proceso de validación de requisitos y tests que constaten que las funcionalidades cumplen lo especificado.

Trello

Trello [25] es una aplicación web que nos permite crear y administrar paneles Kanban [26]. Su uso durante el proyecto ha sido especialmente sencillo debido a que se trata de una de las herramientas que BitBucket integra en su consola web.

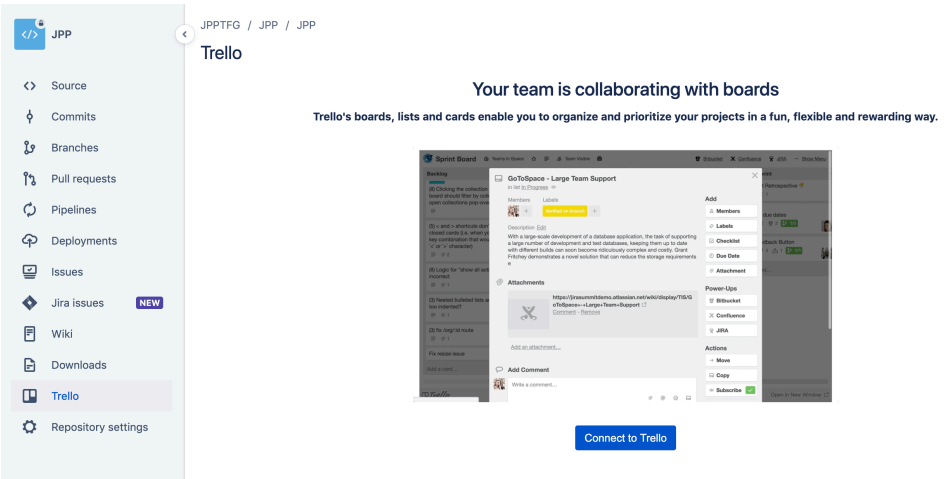

Ilustración 26: Panel de Atlassian Bitbucket, sección de Trello

Estimación temporal

Tras listar y categorizar los distintos requisitos funcionales y no funcionales de la aplicación (en BitBucket, con la ayuda del panel Kanban, etc.) se realizó una estimación previa para el proyecto. En esta estimación, se calculó que el proyecto tendría una carga de trabajo de aproximadamente 96h.

Finalmente, tras trabajar en la entrega de 6 Hitos distintos que englobaban sprints de 2 semanas cada uno, se ha obtenido que los requerimientos más prioritarios e iniciales fueron desarrollados en 61h.

No obstante, durante el desarrollo, se fueron introduciendo en la planificación nuevas tareas que han ido surgiendo.

Estas tareas fueron estimadas en un total de 38h e incluían aspectos menos críticos como la autenticación por redes sociales o la optimización de la carga de recursos por medio del Lazy Load (que se explicará más adelante).

Las nuevas tarjetas fueron añadidas al Backlog y entregadas durante los Hitos 7 y 8, que supusieron un coste final de 41h más de trabajo.

En conclusión, la estimación temporal de la aplicación se resumiría en 134h para las cuales se han utilizado finalmente 102h. La diferencia puede resultar notable, es interesante anotar que la razón por la que se ven tan descompensados los números no es otra que el uso de Firebase como BBDD. Al inicio se supuso mucho más costosa y compleja la implementación de esa parte del Back-End por falta de contexto y desconocimiento inicial de la tecnología, lo que hizo que solo en la evaluación de esa tarea, ya hubiese un desencuentro de 16h.

Los detalles del resto de las estimaciones pueden consultarse en el tablero del repositorio en bitbucket.

Ejemplo práctico sobre la metodología aplicada

Para entender mejor este proceso, se ha analizado un ejemplo práctico, por el cual comenzaríamos el flujo definiendo varias tareas en Trello:

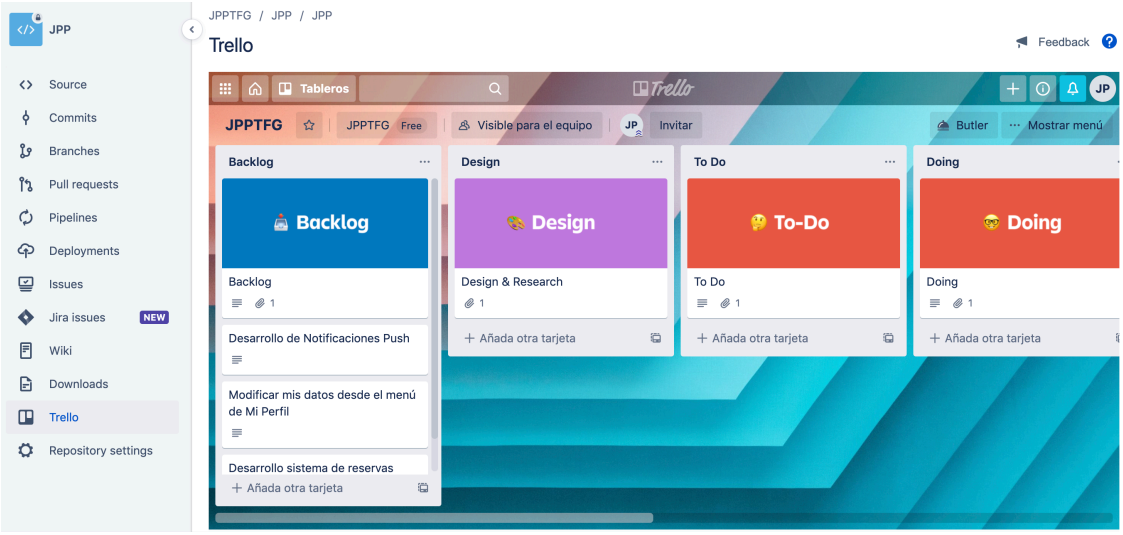

Ilustración 27: Panel de Bitbucket, Tablero de Trello

Y en cada detalle, especificaremos:

- Una checklist con los subapartados de los que pueda constar el requisito junto a una breve descripción de cada uno.
- Una estimación temporal y el tiempo incurrido en el desarrollo de la tarea.

Trello

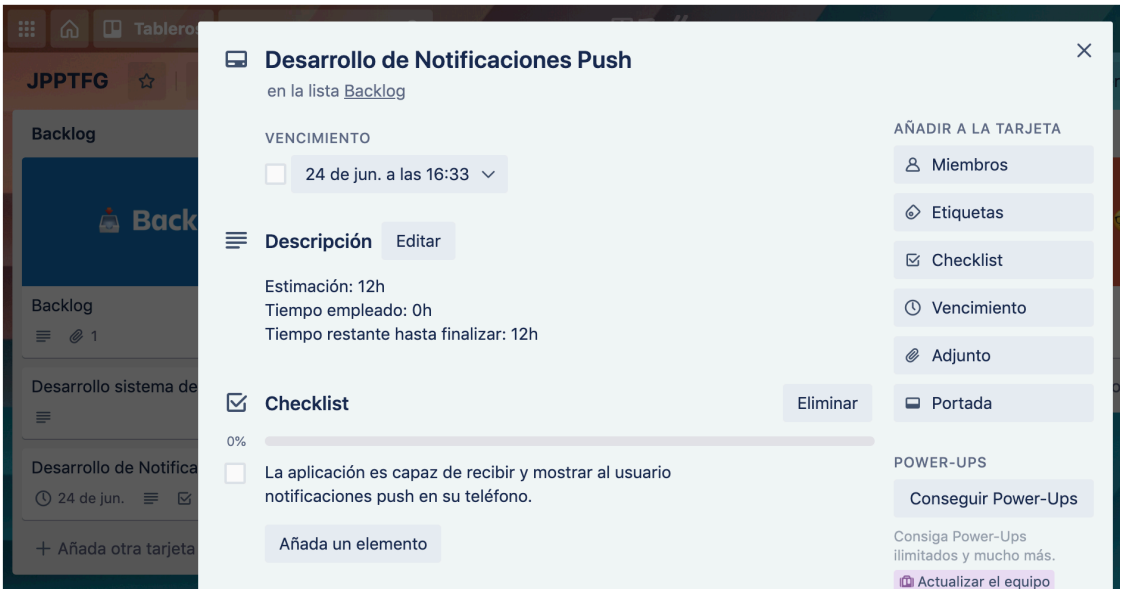

Ilustración 28: Tarjeta de Trello

Una vez se decide qué tareas vamos a implementar para la siguiente entrega (Release o Hito), movemos las tarjetas indicadas al apartado TO DO y le asignamos una fecha de vencimiento (la próxima entrega del incremental).

Cuando un desarrollador se asigna la tarea y la pasa a “Doing”, crea una rama en Sourcetree con el nombre identificador de la release a la que pertenece y una descripción de la tarea que va a llevarse a cabo en dicha rama.

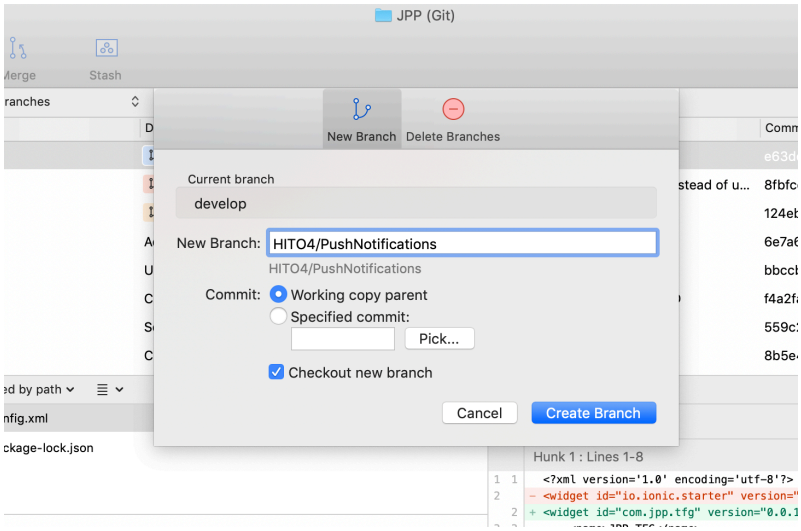

Ilustración 29: Creando una rama en Sourcetree

Las ramas de desarrollo nuevas surgen siempre desde develop.

Conforme se avanza en su implementación, se va actualizando la tarea con comentarios relevantes:

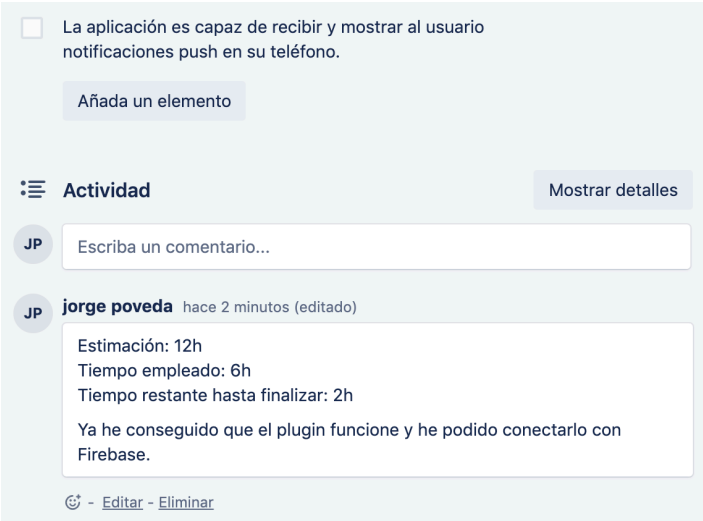

Ilustración 30: Actualizando tarjeta de Trello

Una vez finalizado el desarrollo, hacemos un commit utilizando Sourcetree para guardar los cambios en nuestro repositorio:

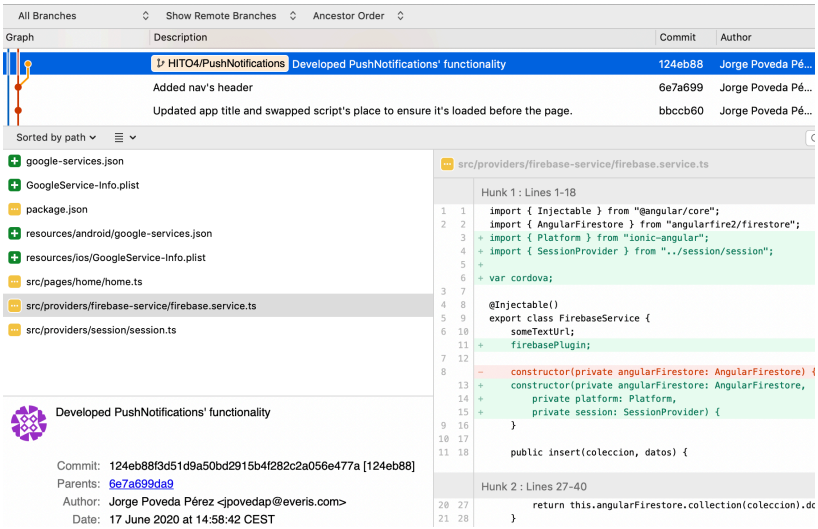

Ilustración 31: Panel de Sourcetree, sección de Historial

Acto seguido, generamos una versión en la que validamos la solución:

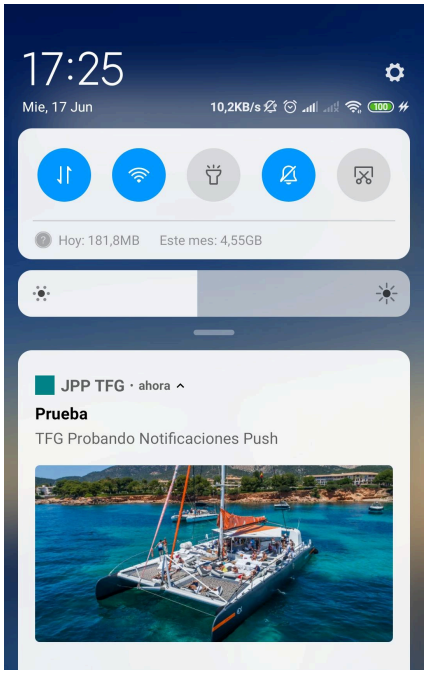

Ilustración 32: Captura de una notificación push

Luego, vamos a Sourcetree de nuevo y hacemos clic derecho sobre la rama develop y pulsamos en “Create Pull Request”. Esta opción nos abre en el navegador:

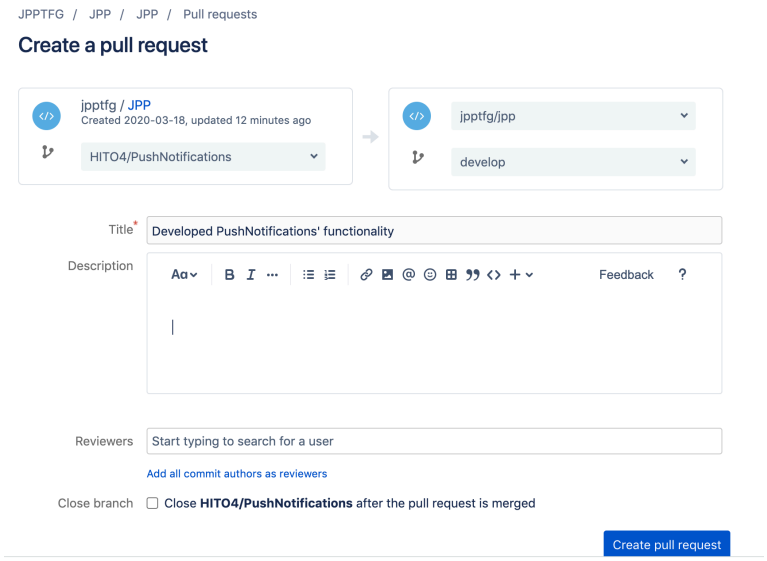

Ilustración 33: Creando un Pull Request a develop

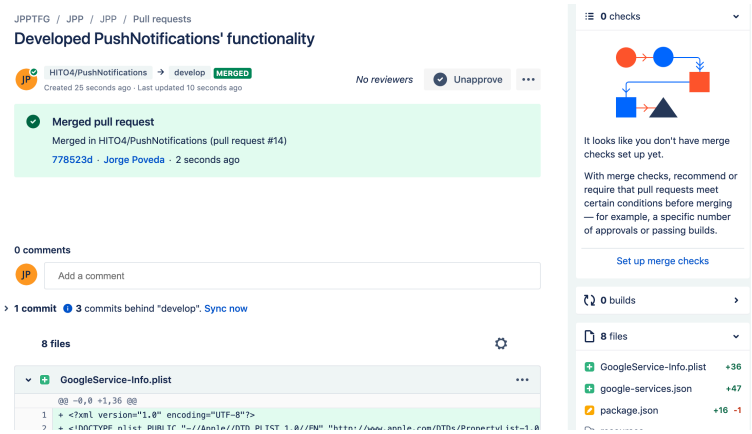

Ilustración 34: PR aprobado y mergeado

De este modo, podemos crear el PR, aprobarlo (nosotros mismos, ya que no hay revisor) y realizar un “merge” a continuación, obteniendo una versión de la aplicación con funcionalidad incrementada.

Repetimos este proceso con el resto de las tareas del Hito y cuando hayamos completado todas, generamos una versión desde develop para hacer un testeo completo de todas las funciones añadidas.

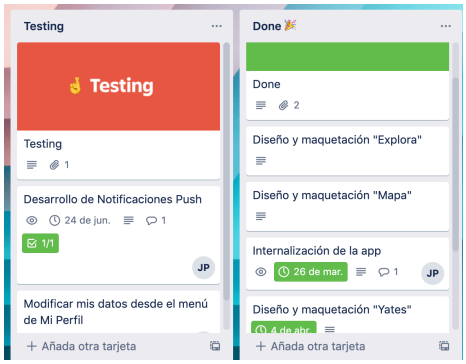

Ilustración 35: Panel Kanban de Trello

Cuando hayamos finalizado las pruebas de forma satisfactoria, llevamos las tarjetas a Done y las damos por completadas.

El paso final de este proceso ocurre al realizar la entrega. En este momento, seria el “cliente” quien recibe la versión con la que se ha validado la nueva funcionalidad y cuando ellos hagan lo propio, se actualizará la rama estable (master).

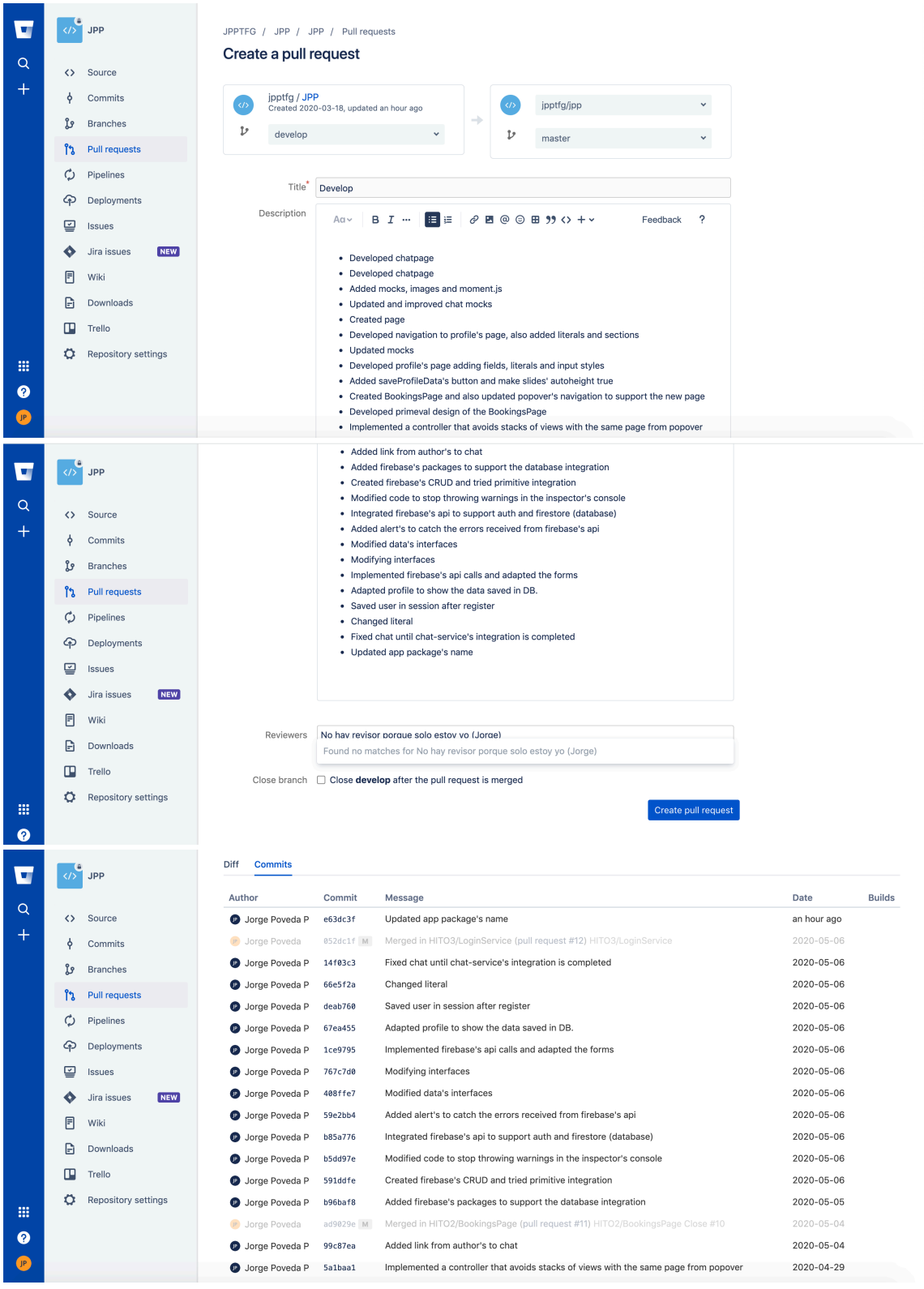

Ilustración 36: Tres figuras que muestran el mergeo de develop a la rama estable

Dando por terminado el ciclo de desarrollo.

REQUISITOS

Se ha creado un listado con los distintos requisitos de la aplicación.

Funcionales

Que detallan las funcionalidades básicas del producto a alto nivel:

- Debe existir un sistema de autenticación que verifique la identidad del usuario ya sea desde un navegador, dispositivo Android o iOS.
- El software debe ser capaz de registrar un usuario, guardando su nombre completo, teléfono, email de contacto, idioma predeterminado, nombre de usuario y contraseña.
- Un usuario registrado previamente, puede autenticarse en la aplicación utilizando las credenciales con las que lo hizo.
- La aplicación tiene que internacionalizarse tanto en inglés como en castellano.
- Existen varios roles de usuario que diferencian distintas capacidades.
- El usuario debe ser capaz de modificar sus datos personales o añadir una foto de perfil desde la aplicación.
- El software posee una pantalla Explora con los resultados más novedosos o relevantes de entre las ofertas disponibles.
- Posee una pantalla de Destinos con un listado de distintas localizaciones para filtrar las ofertas.
- Posee una pantalla de Embarcaciones con un listado de las distintas embarcaciones disponibles para su filtrado.
- Hace uso de la API externa de Google Maps para mostrar las ofertas relevantes en un mapa interactivo.
- Se puede enviar mensajes entre usuarios y/o Soporte (Administrador).
- Un usuario debe poder cerrar su sesión tras haberse identificado.
- Un usuario puede recibir notificaciones push en su dispositivo. También debe ser capaz de poder decidir qué tipos de alertas quiere recibir.
- Dispone de un apartado Mis Reservas donde el usuario es capaz de ver un listado con sus reservas activas y el historial previo.
- El software permite la autenticación de un usuario a través de redes sociales.

Al existir diferentes perfiles en la aplicación encontramos funcionalidades que solo pertenecen a ciertos usuarios. Todos aquellos usuarios que han sido identificados por el administrador como usuarios ofertadores, además de todos los requisitos anteriores, la aplicación para ellos también:

- Dispone de una pantalla para crear una oferta nueva en la que se pueden configurar campos como el titulo de la oferta, descripción, número de pasajeros, precio, fechas relevantes, lugares de origen y destino, así como una foto de la embarcación y su tipo.
- Se muestra en una sección de Mis Reservas el listado de ofertas publicadas por el usuario.

En el rol más completo, encontramos al administrador, a quien la aplicación debe permitir:

- Añadir nuevos destinos y tipos de embarcaciones.
- Responder mediante el Chat a los mensajes que los usuarios envían a “Soporte”.

No Funcionales

Que especifican los criterios de calidad impuestos al software:

- Los datos sensibles deben almacenarse encriptados usando un algoritmo seguro.
- La transmisión de datos entre Front y Back-End debe ser segura y encriptada.
- Todos los datos guardados en un dispositivo físico deben haber sido encriptados.
- Las peticiones a Base de datos deben tener un tiempo de respuesta corto y que no lastre la experiencia de usuario.
- Los momentos durante los cuales el usuario no puede interactuar porque la app esta cargando recursos, tanto al iniciarse como entre navegaciones, deben ser mínimos.
- La interfaz de usuario es agradable a la vista e intuitiva, haciendo que la experiencia de uso sea atractiva, sencilla y rápida.
- La vista de la aplicación es responsiva y demuestra comportarse sin fallos en plataformas y dispositivos de diversos tamaños.
- El producto se desarrolla siguiendo el patrón convencional marcado por el lenguaje, haciendo uso de buenas prácticas que garanticen su escalabilidad, extensibilidad y rendimiento.
- La BBDD debe ser capaz de soportar una concurrencia simultánea de 500 usuarios.

DISEÑO

Pantallas

La primera vista que nos encontramos al iniciar la EmbarcApp es la pantalla de identificación. En ella podemos elegir registrarnos, identificarnos con un mail si previamente lo hemos hecho o acceder mediante login con redes sociales.

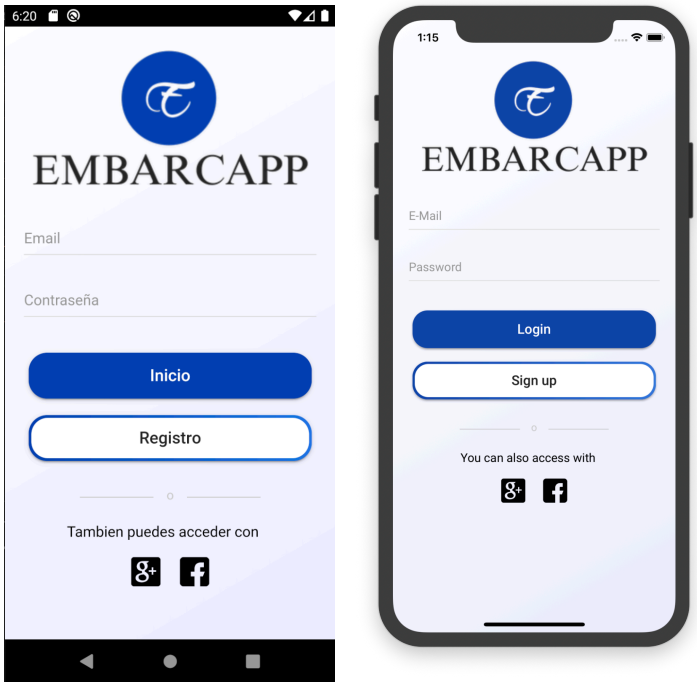

Ilustración 37: Captura de la pantalla de inicio en un dispositivo Android en castellano(1) y en un simulador iOS en inglés (2)

En el caso de querer registrarnos, podríamos realizar el proceso en la pantalla de Registro:

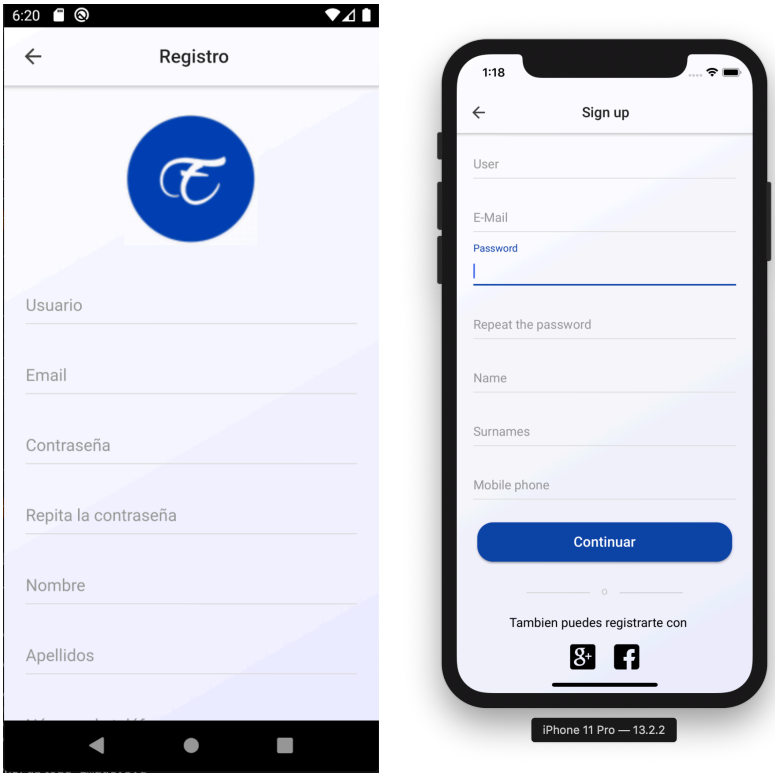

Ilustración 38: Captura del proceso de Registro en la aplicación en un dispositivo Android (1) e iOS (2)

Tras autenticarnos correctamente, la pantalla de inicio seria “Explora”. Donde veríamos un listado de ofertas con detalles relevantes sobre las mismas. En esta misma pantalla, es donde se aplicarían los filtros de destinos, barcos, etc…

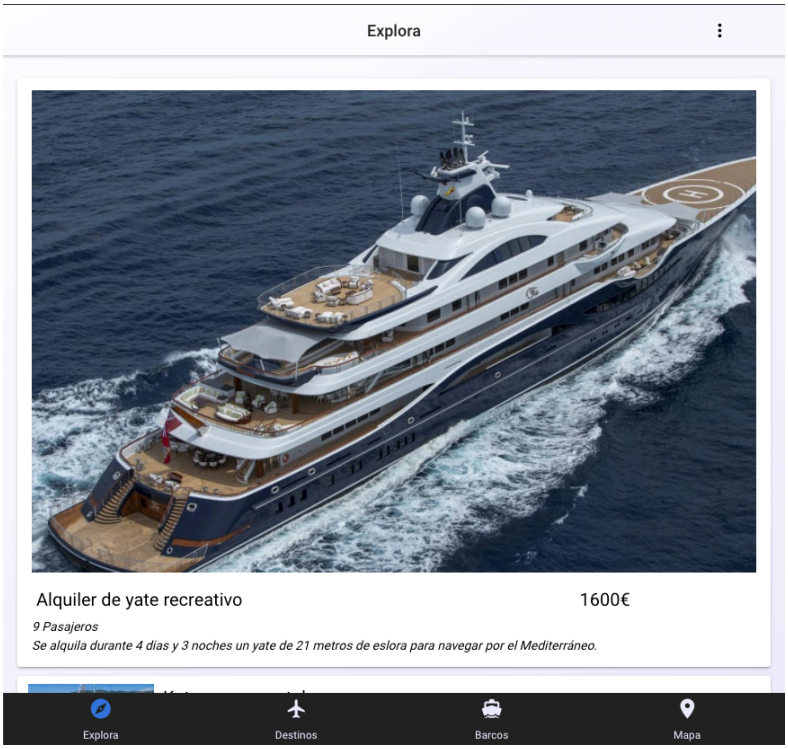

Ilustración 39: Pantalla Explora en una PWA

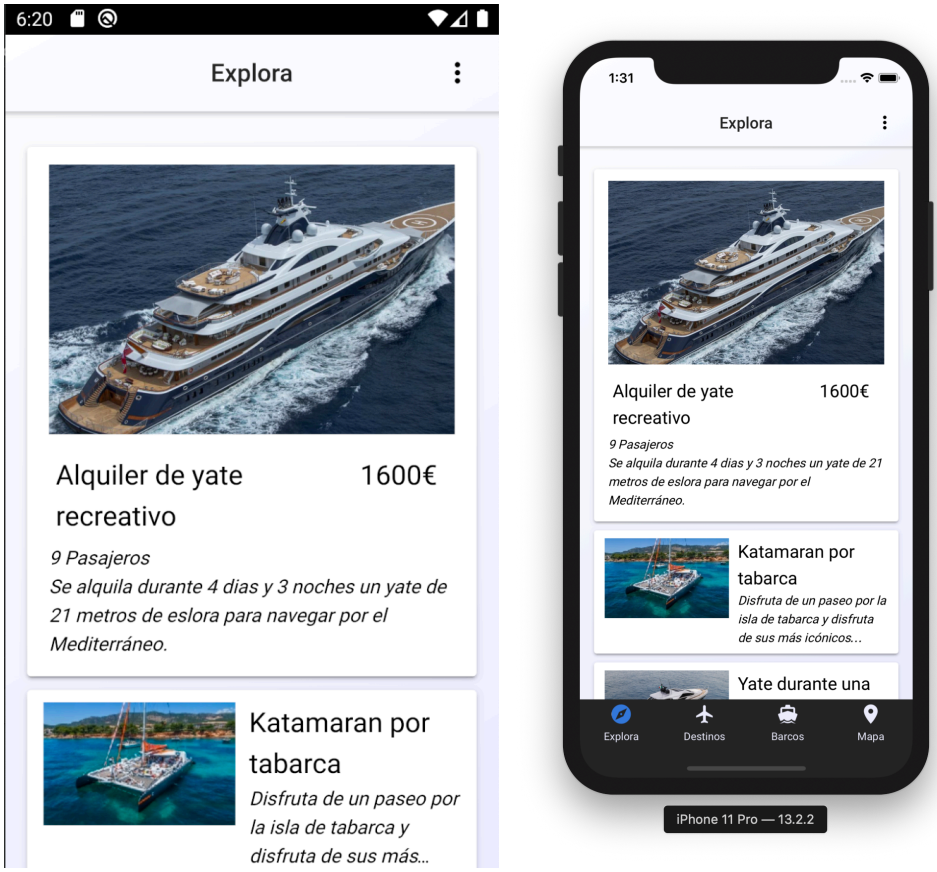

Ilustración 40: Pantalla Explora en un dispositivo Android (1) e iOS (2)

Para ver más detalles sobre la oferta o reservarla, tendríamos que hacer click. La siguiente vista sería entonces:

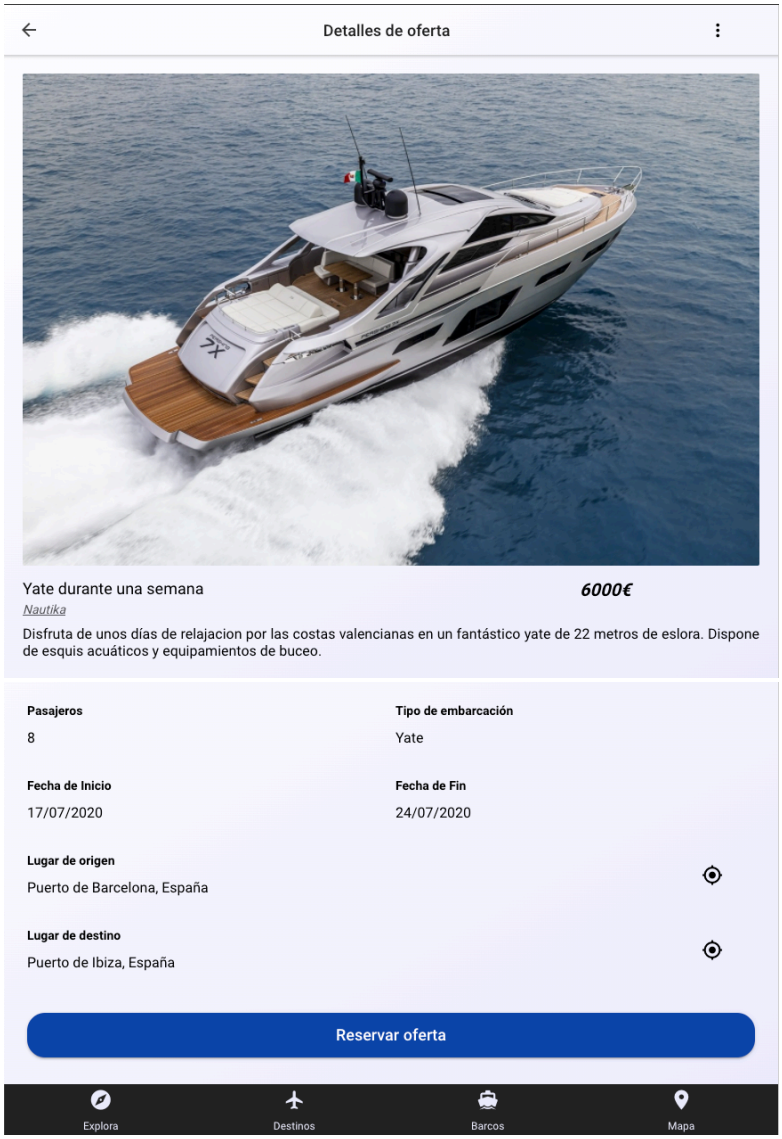

Ilustración 41: Pantalla Detalles de Oferta en PWA

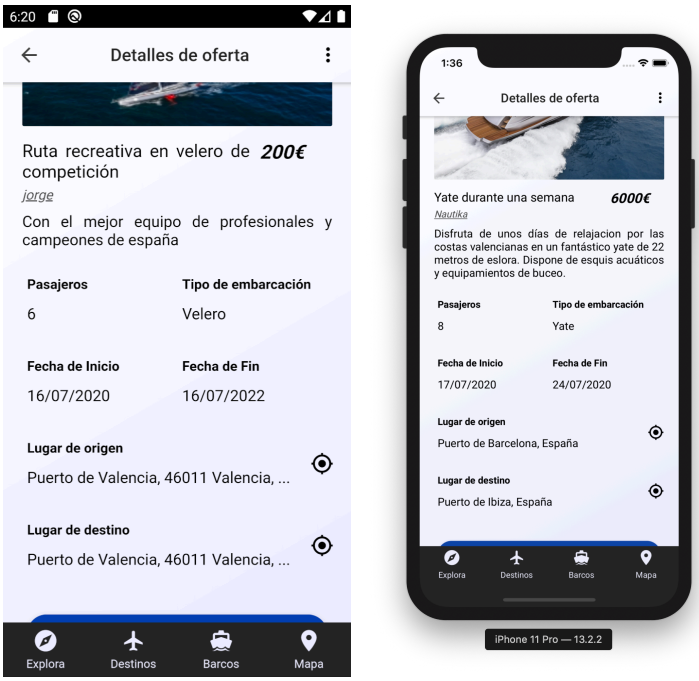

Ilustración 42: Pantalla Detalles de Oferta en un dispositivo Android (1) e iOS (2)

En esta pantalla también podríamos ir la de Mapas pulsando el icono que hay al lado de los lugares de origen y destino. De este modo, el usuario puede ver mejor la localización.

En la sección de Destinos encontramos imágenes atractivas de lugares donde se puede navegar. Estos destinos tienen asociados geomarcadores que nos permiten filtrar las búsquedas por cercanía.

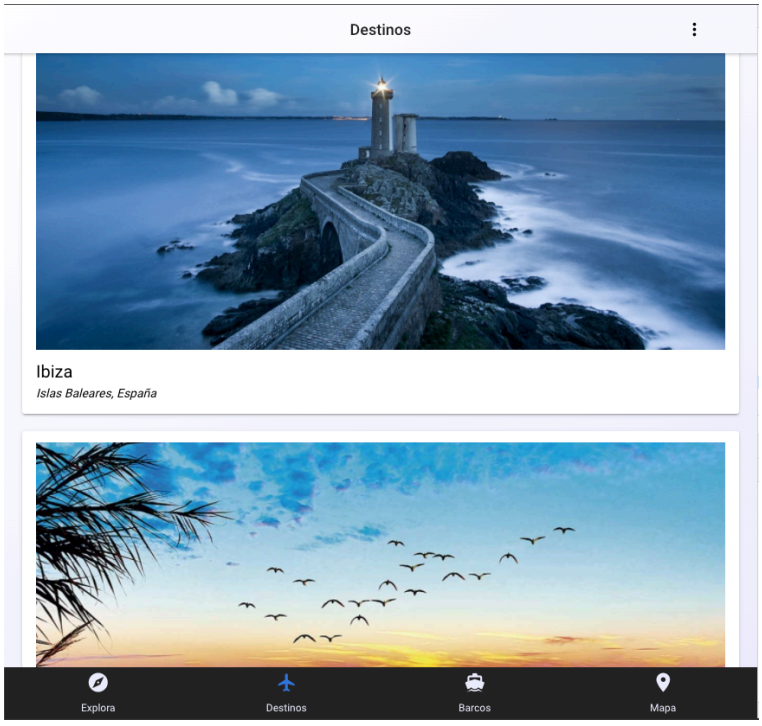

Ilustración 43: Pantalla de Destinos en una PWA

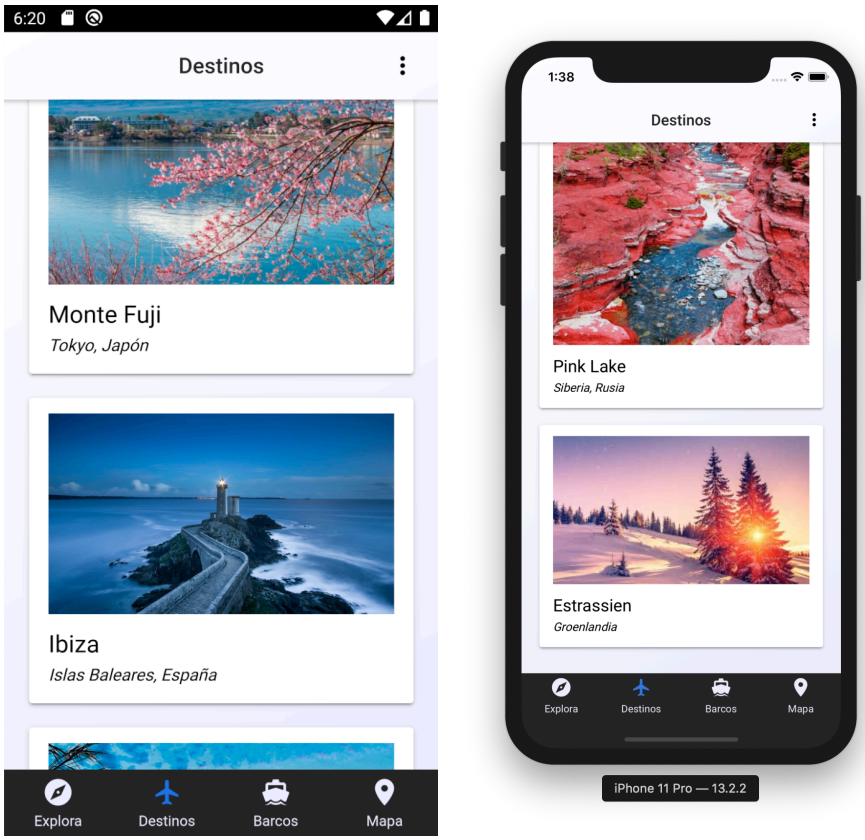

Ilustración 44: Pantalla de Destinos en un dispositivo Android (1) e iOS (2)

En la sección Barcos podríamos ver un listado interactivo con imágenes de los distintos tipos de embarcaciones. Seleccionando una, podríamos filtrar las ofertas para que solo aparezcan las que tengan como embarcación, una de ese tipo:

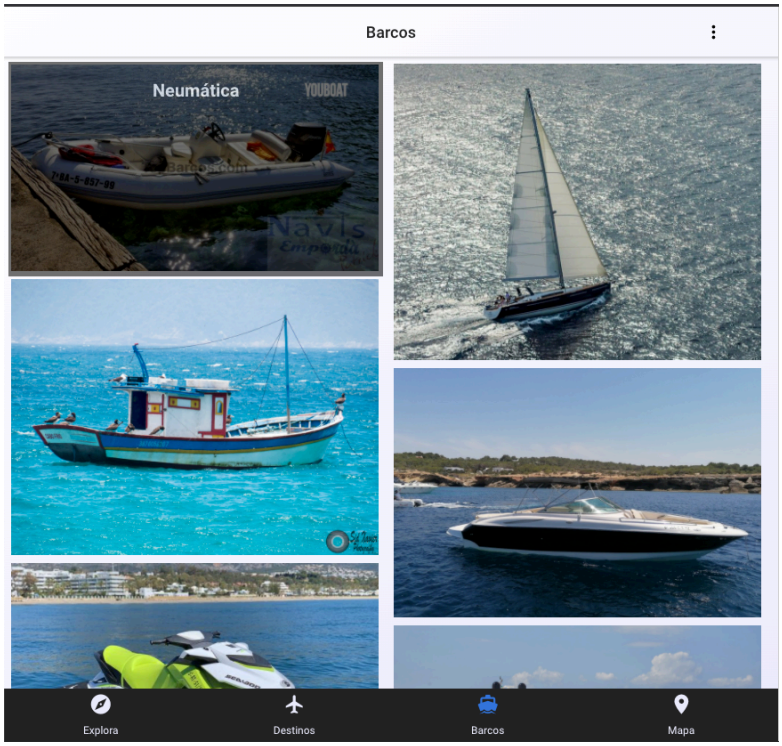

Ilustración 45: Pantalla de Barcos en una PWA con idioma castellano

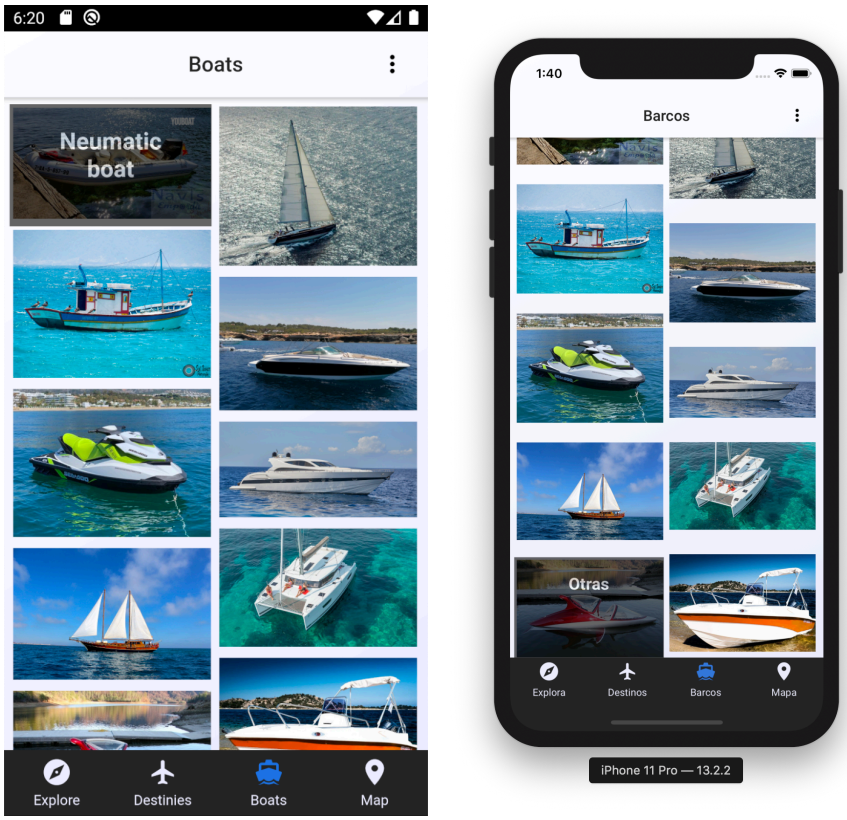

Ilustración 46: Pantalla Barcos en un dispositivo Android (1) con Inglés e iOS (2) en castellano

En el Mapa tenemos marcadores de todas las ofertas disponibles.

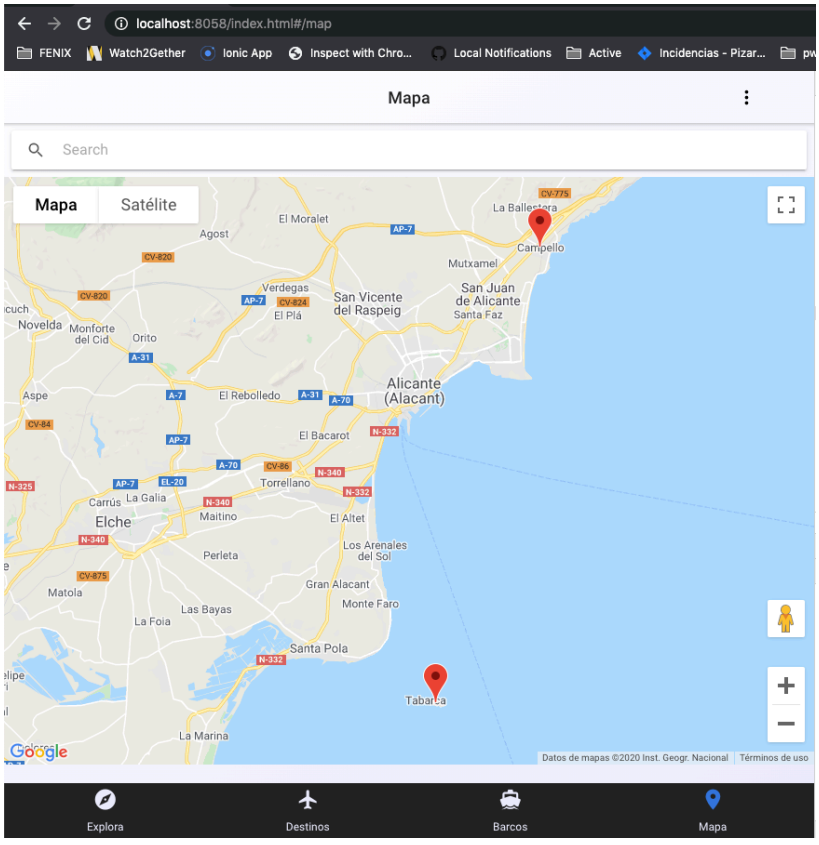

Ilustración 47: Pantalla de Mapa en una PWA con idioma castellano

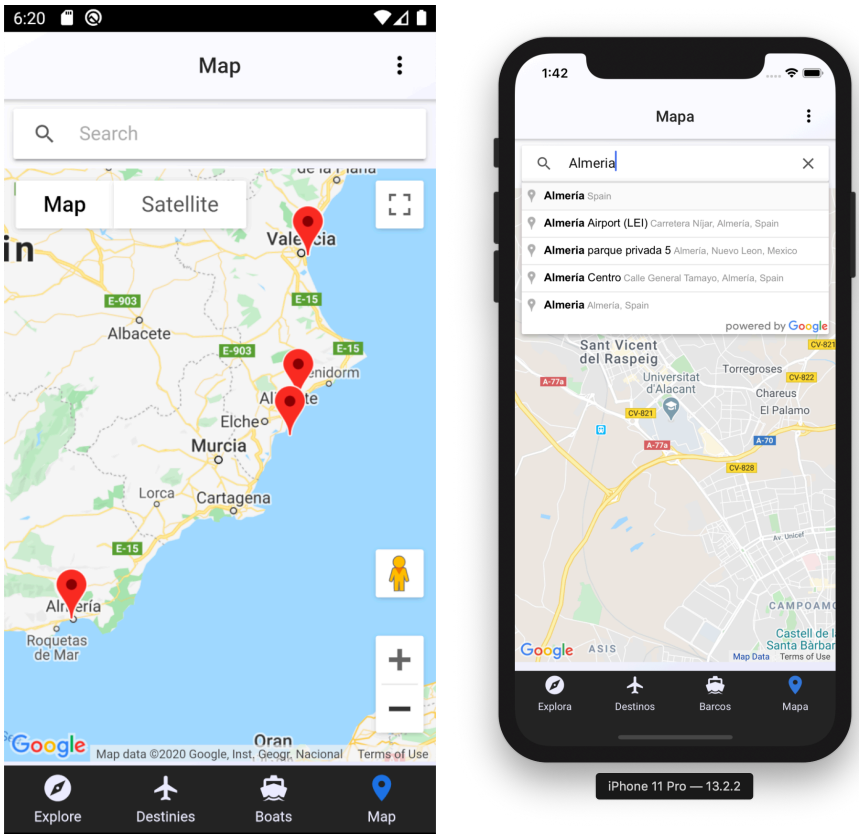

Ilustración 48: Pantalla Mapa en un Android (1) e iOS (2)

Además del Menú en el “Footer” (que se encuentra en la parte baja de la pantalla) , existe otro con más funcionalidades al que se accede desde el icono con 3 puntos de la parte superior derecha.

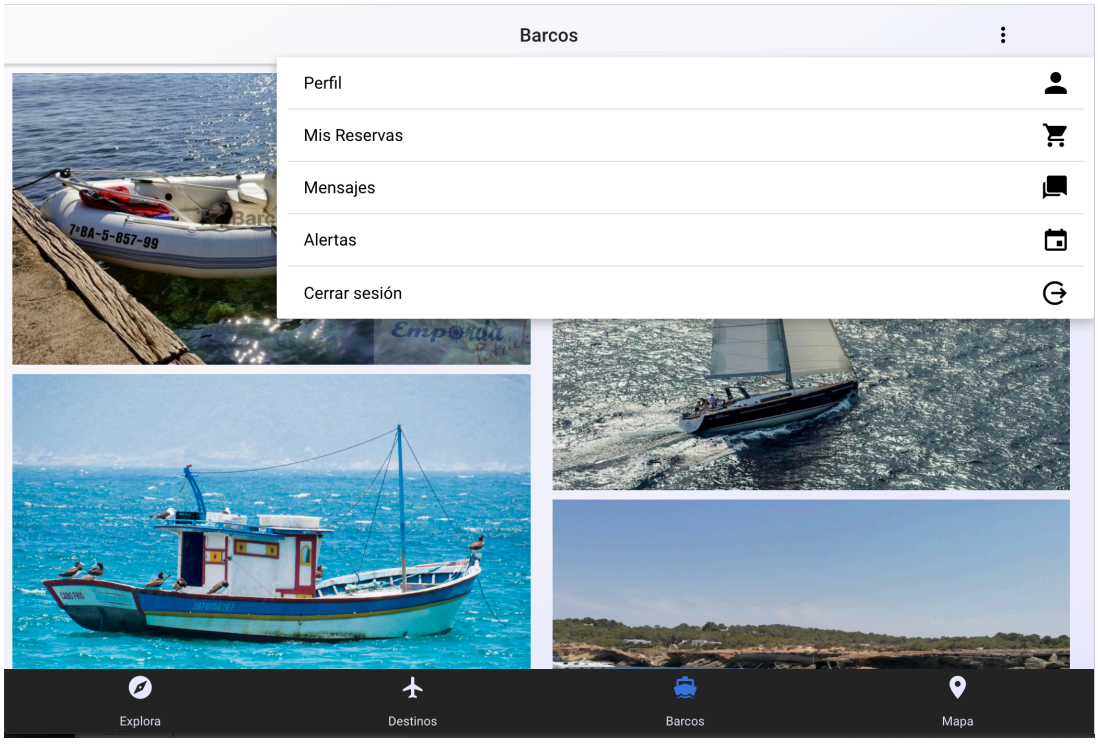

Ilustración 49: Menú contextual superior en PWA con un usuario normal

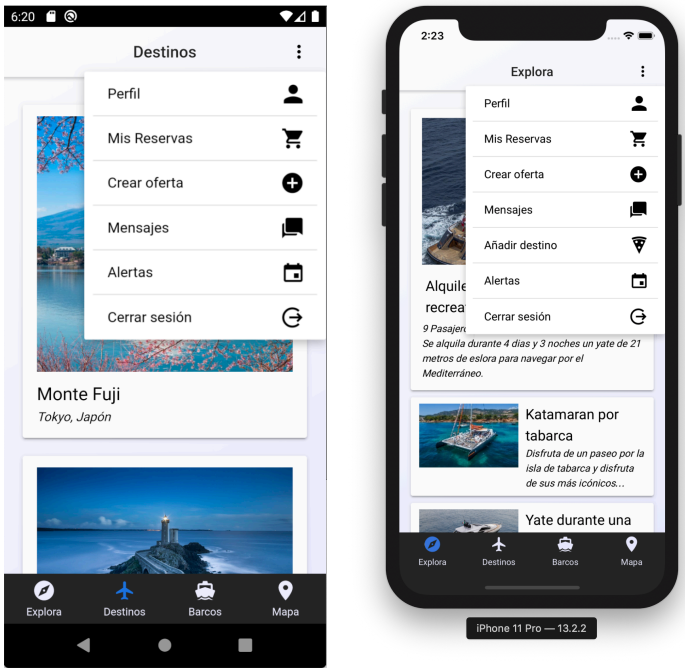

Ilustración 50: Menú contextual superior con un usuario ofertador en Android (1) y con un administrador en iOS (2)

La primera sección del Menú es Mi Perfil. En esta pantalla podemos modificar nuestros datos, así como la imagen de perfil pulsando sobre ella. Hay campos como el correo o el nombre de usuario que no pueden modificarse, por lo que aparecerán como desactivados.

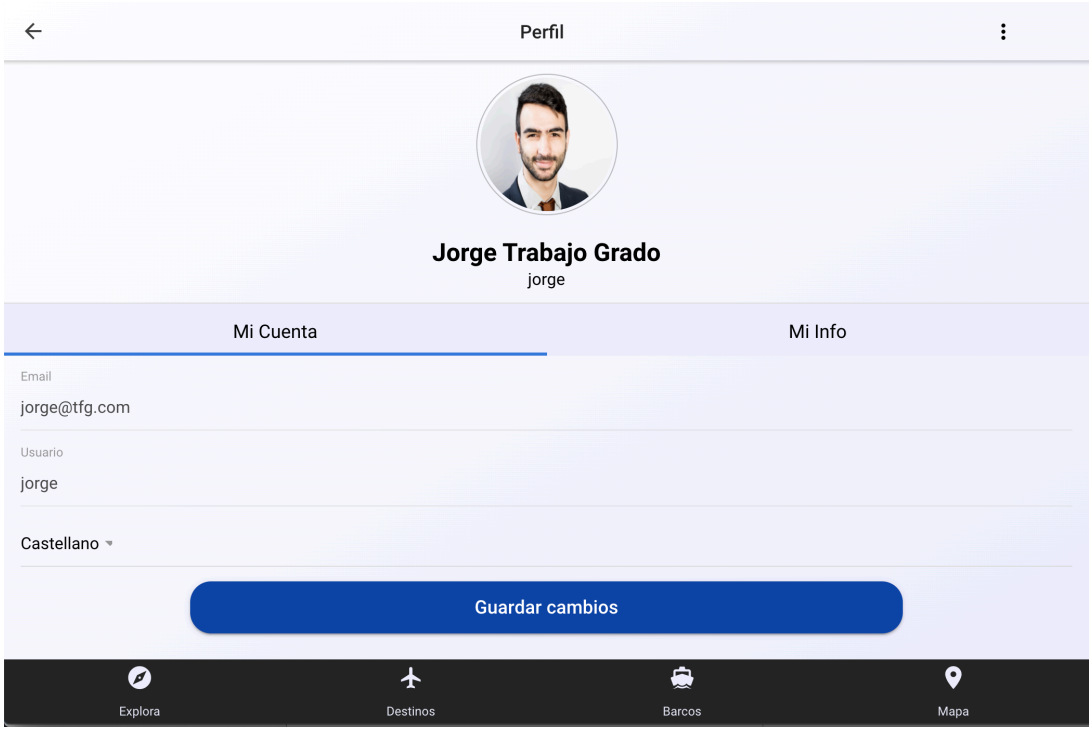

Ilustración 51: Pantalla de Perfil en una PWA con idioma castellano

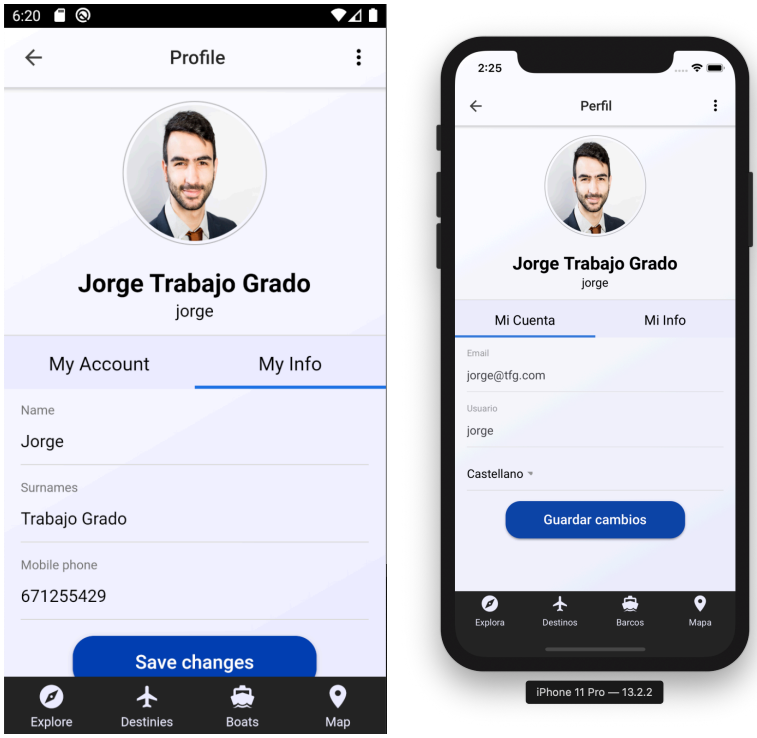

Ilustración 52: Pantalla Perfil en Android (1) e iOS (2)

En Mis Reservas encontraríamos un listado con las reservas activas y el historial del usuario:

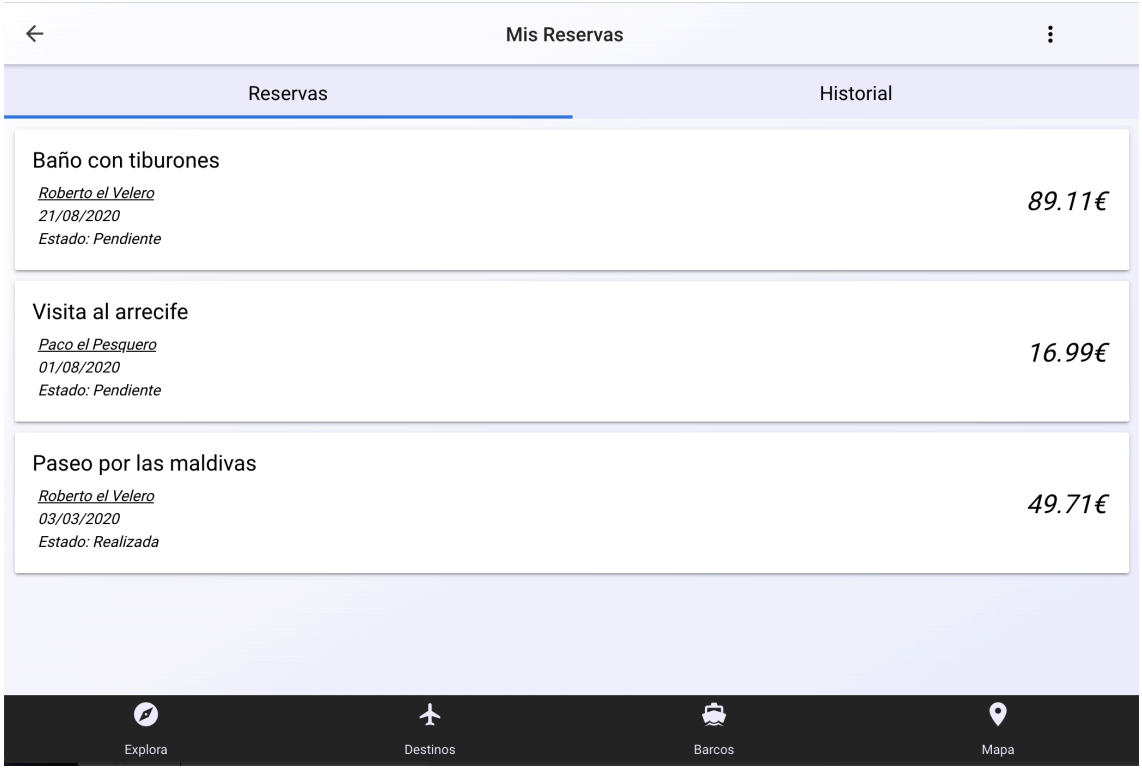

Ilustración 53: Pantalla Mis Reservas en una PWA

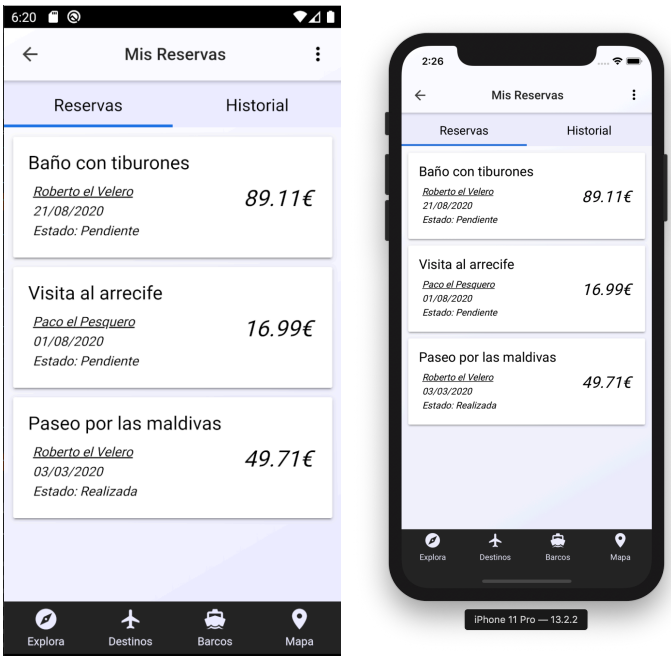

Ilustración 54: Pantalla Mis Reservas en Android (1) e iOS (2)

La sección de Crear Oferta nos permite añadir una nueva. Para hacerlo tendremos que rellenar los campos, añadir imágenes y marcar su localización en el mapa.

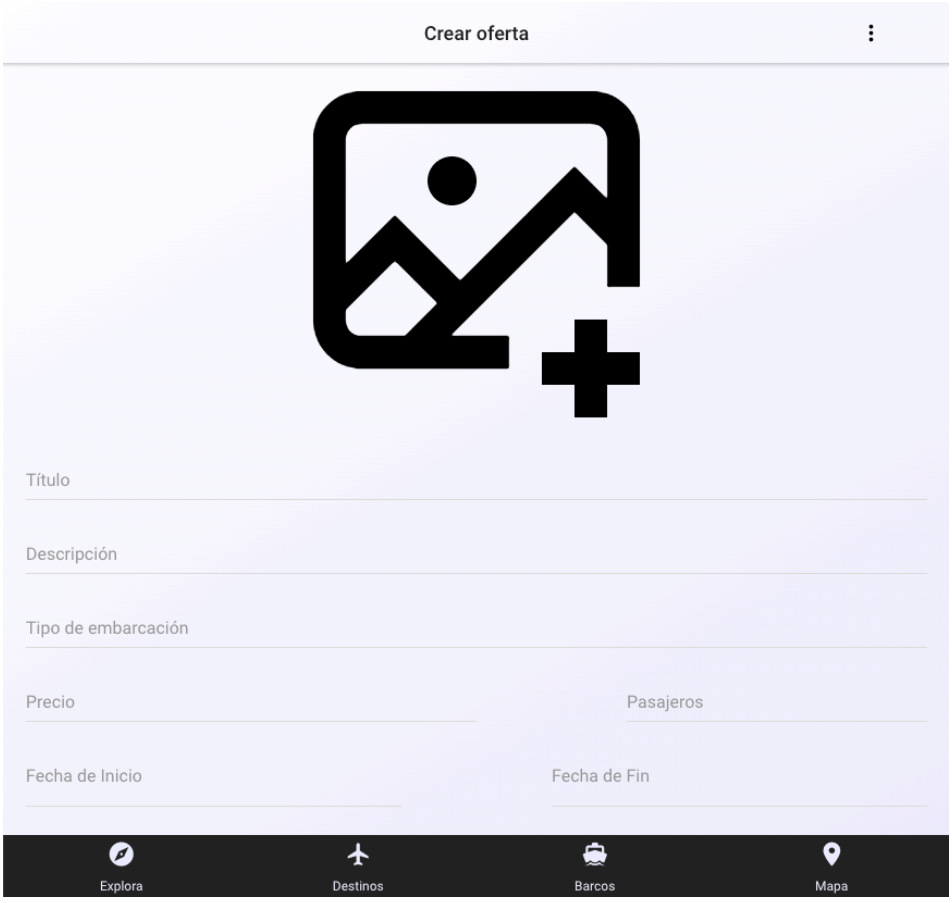

Ilustración 55: Pantalla de Crear Oferta en PWA

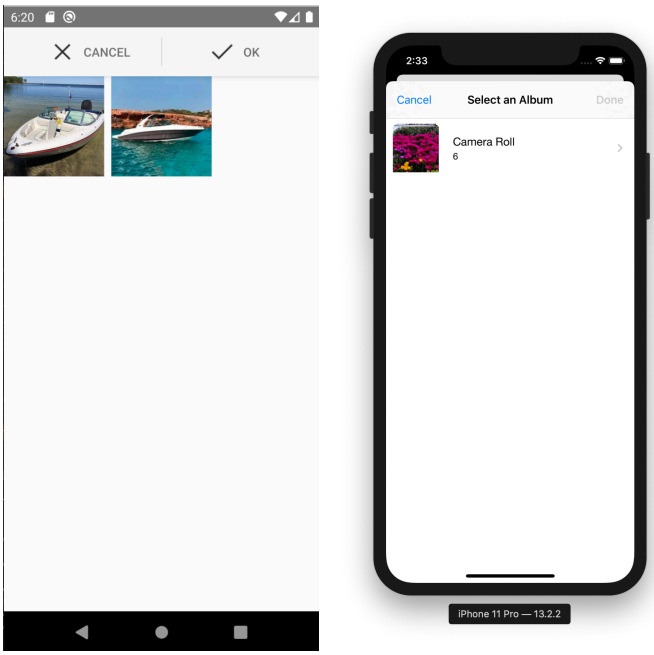

Ilustración 56: Plugin de acceso a la galeria en Android (1) e iOS (2)

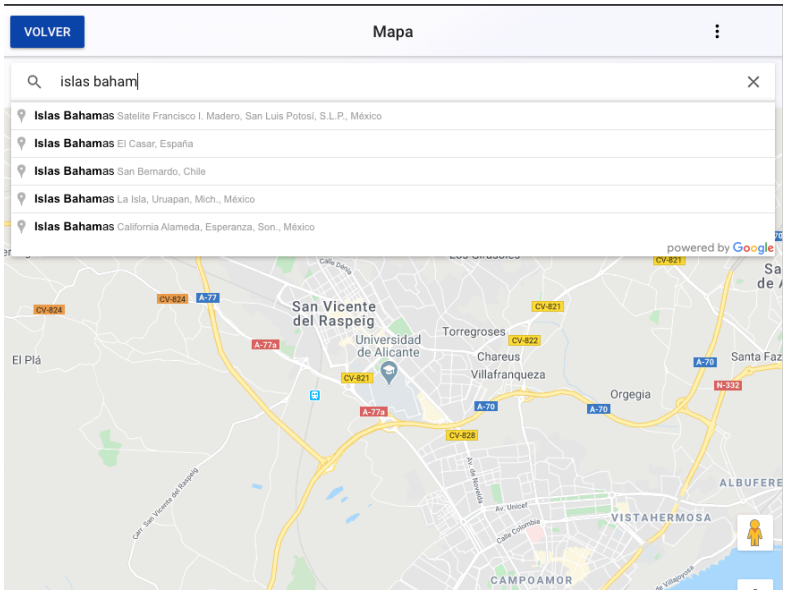

Ilustración 57: Pantalla de Mapa, usando el buscador para marcar una localización en una PWA

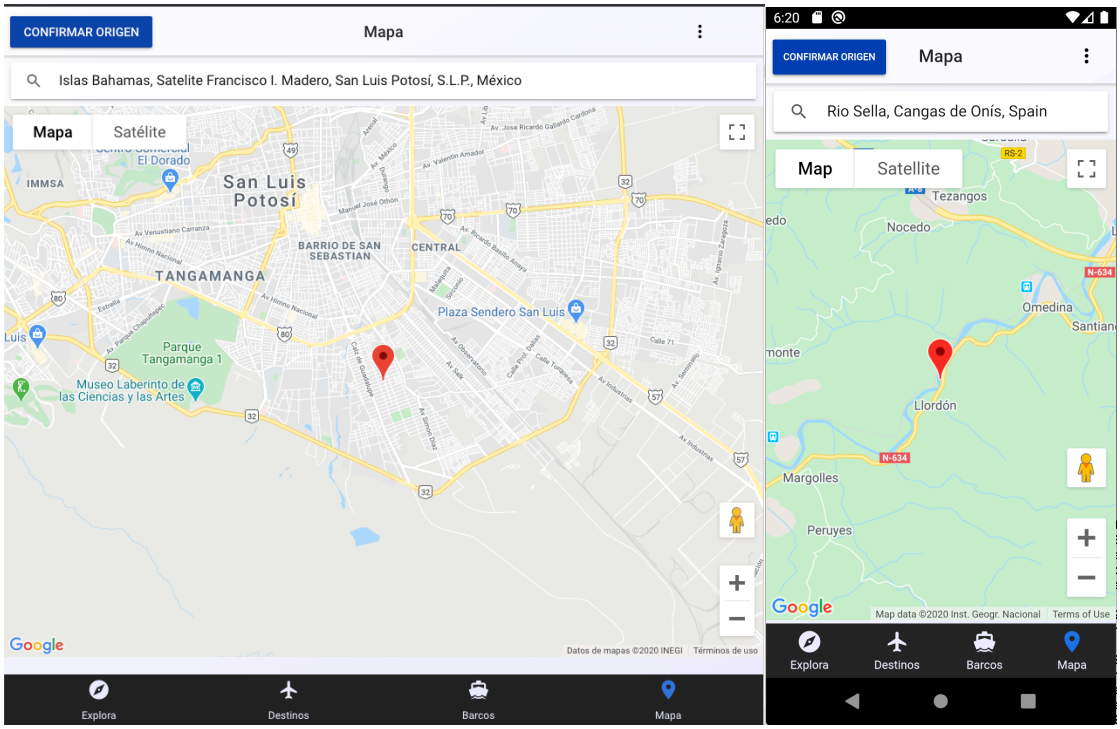

Ilustración 58: Pantalla de Mapa tras encontrar y marcar la localización buscada en PWA (1) y en Android (2)

Una vez seleccionadas las localizaciones, y rellenados los datos, ya tendríamos nuestra oferta.

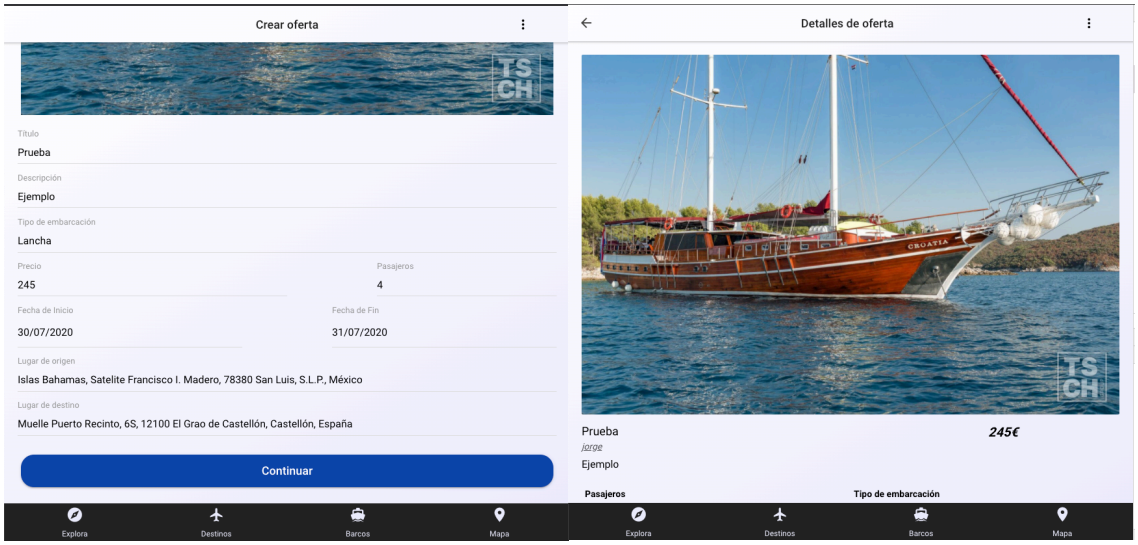

Ilustración 59: Pantalla de Crear Oferta en PWA (1 y 2)

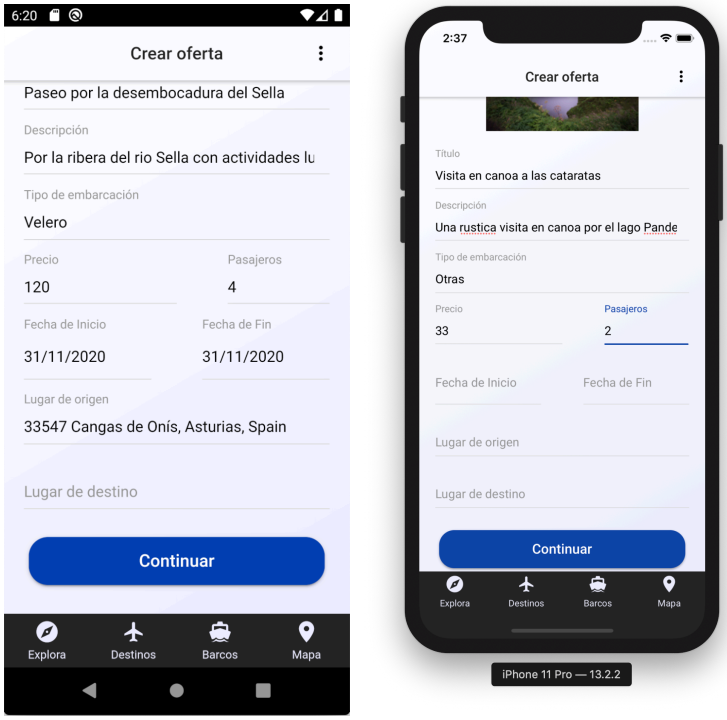

Ilustración 60: Pantalla de Crear Oferta en Android (1) e iOS (2)

En la página de Alertas podremos configurar qué notificaciones push queremos recibir:

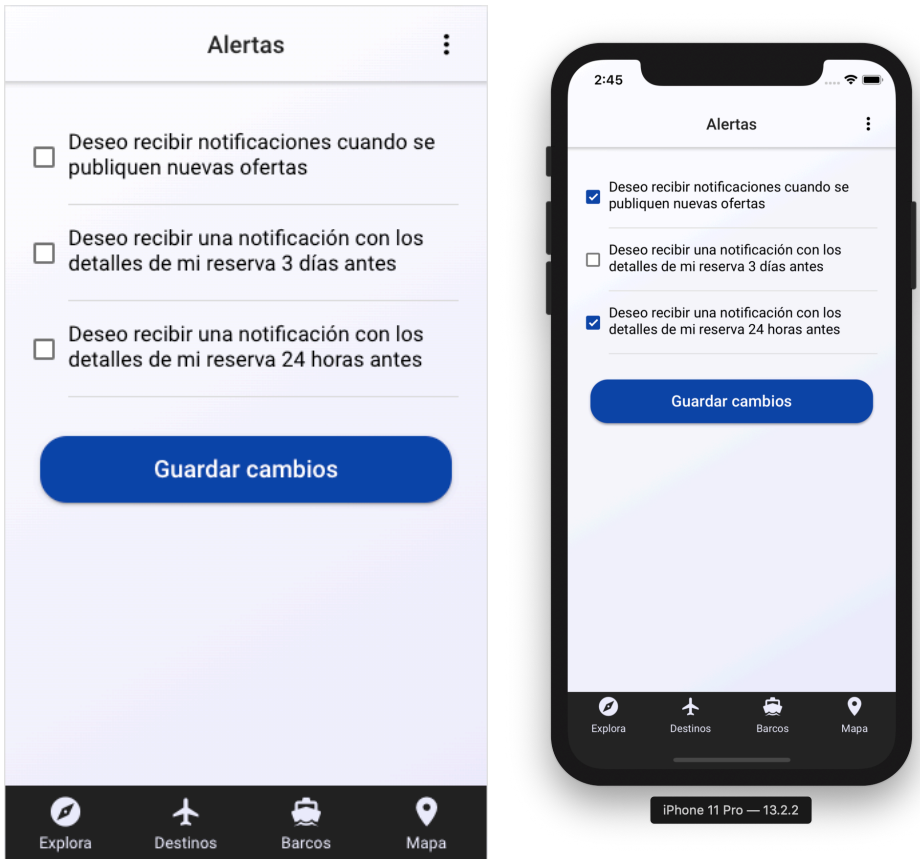

Ilustración 61: Pantalla Alertas en Android (1) e iOS (2)

Huelga decir que independientemente de la configuración de notificaciones de la cuenta, nunca nos llegarán notificaciones si no le damos a la aplicación los permisos correspondientes:

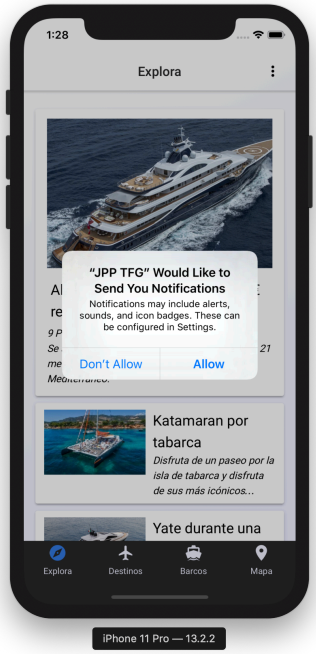

Ilustración 62: Pantalla de Explora con una alerta de iOS sobre la aceptación de Notificaciones Push

La aplicación también dispone de una sección de mensajería. Incorpora un chat en tiempo real ideado para contactar con otros usuarios y el administrador o miembros de "Soporte".

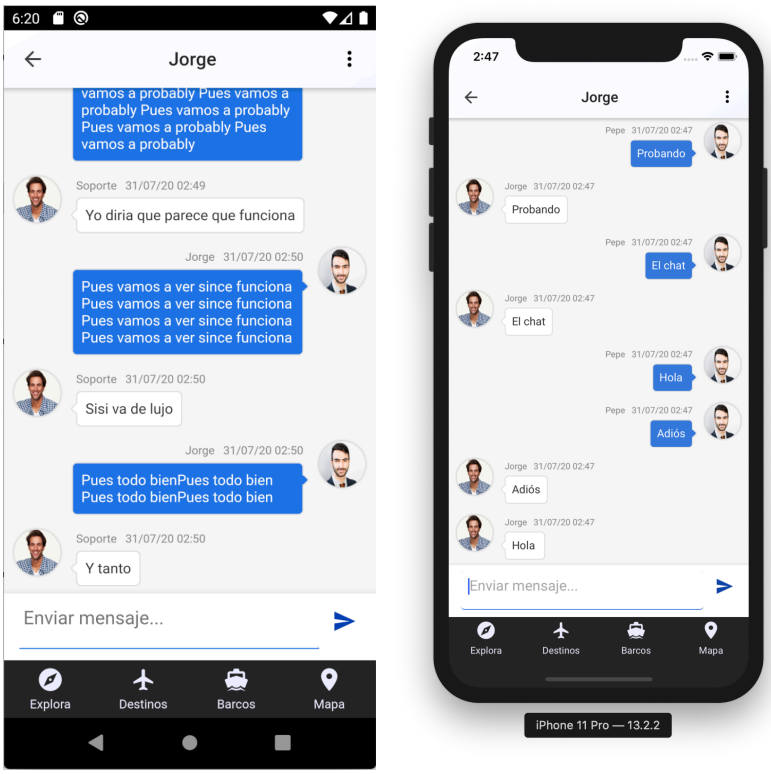

Ilustración 63: Pantalla de Chat en un dispositivo Android (1) e iOS (2)

Y por ultimo la de Añadir destino, a la cual solo podría acceder el administrador. Al añadir un destino, lo veremos aparecer en la sección de Destinos:

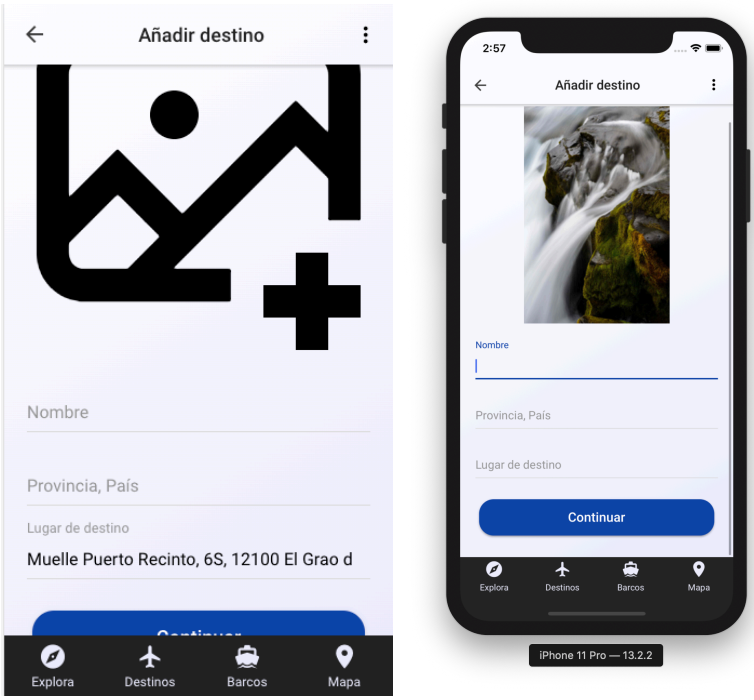

Ilustración 64: Pantalla Añadir Destino con un Android (1) y un iOS (2)

ARQUITECTURA

El Framework que define la arquitectura de la aplicación es Angular.

Angular como tal no tiene un modelo-vista-controlador (MVC) clásico, ya que el modelo tiene mucha relación con la vista.

Esta situación se produce debido a una característica fundamental de Angular, “two-way data binding”, que aparece comúnmente en el patrón MVVM (Model-View-ViewModel).

A la hora de sincronizar los datos entre la vista y el modelo-vista encontramos que son totalmente dependientes, lo que significa que en la vista se puede modificar el modelo y viceversa.

La independencia que se produce en un modelo-vista-controlador común no sucede en estas aplicaciones. Si que es cierto que posee muchas características de este patrón, pero no resulta predominante.

Model-View-Whatever

Modelo-Vista-Loquesea se utiliza para definir un patrón en el que realmente se prima la flexibilidad. Angular permite separar cómodamente la lógica de presentación, de su estado o de los objetos de negocio, por lo que ha demostrado que, a pesar de no seguir los patrones convencionales, posee una curva de aprendizaje que no se ve lastrada por no poder enmarcarlo en ningún patrón concreto.

No es de extrañar por tanto que acabemos englobándolo en el MVW, cuya característica fundamental indica que se debe utilizar “whatever works for you” o algo así como “haz lo que sea que te funcione”.

IMPLEMENTACIÓN

La codificación se ve dividida en dos ámbitos.

Front-End

El grueso del proyecto se encuentra en la aplicación móvil y, por lo tanto, en la parte de Front.

El núcleo del desarrollo se encuentra en Ionic, específicamente en Ionic 3. Para crear un nuevo proyecto, lo descargaríamos con NPM y utilizaríamos la línea de comandos y para iniciar nuestra aplicación:

$ npm install -g @ionic/cli

$ ionic start jpptfg blank --type=ionic-angular

En nuestro caso, "blank" crearía un nuevo proyecto en blanco, y la especificación del "type" nos lo configuraría como Ionic 3.

Una vez generado, podríamos crear páginas con:

$ ionic generate <type> <name> [options] → *$ ionic generate page home*

Este comando nos crearía un nuevo directorio en la carpeta pages de nuestro proyecto:

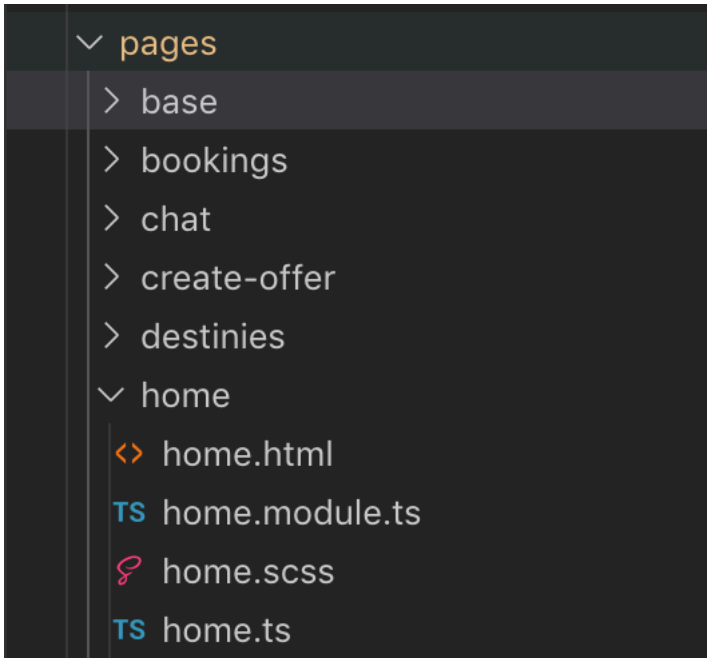

Ilustración 65: Captura del directorio pages donde se aprecia el contenido de los archivos que forman HomePage

En el directorio se encuentran 4 archivos:

- HTML: Donde programamos en HTML5 con la ayuda de notaciones del wrapper (encapsulador) de Ionic (como ion-col, ion-header, o incluso Pipes como el de

traducción…) y Angular (*ngFor, *ngIF…).

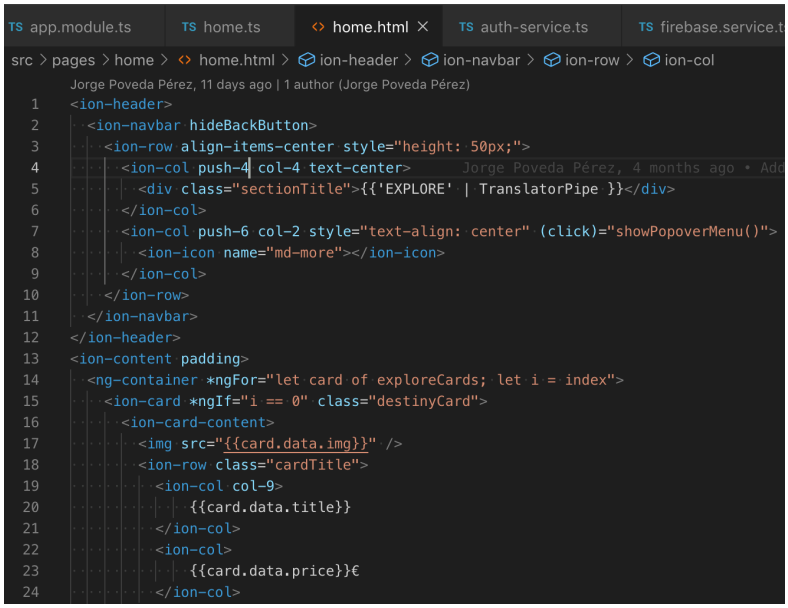

Ilustración 66: Captura del código de home.html.

Los eventos como “click”, “onChange” y demás, provocarían que se llame a métodos de home.ts.

- Module.ts: Donde se declara la página como módulo para que la aplicación sea capaz de importarla e inyectarla con todas sus dependencias.

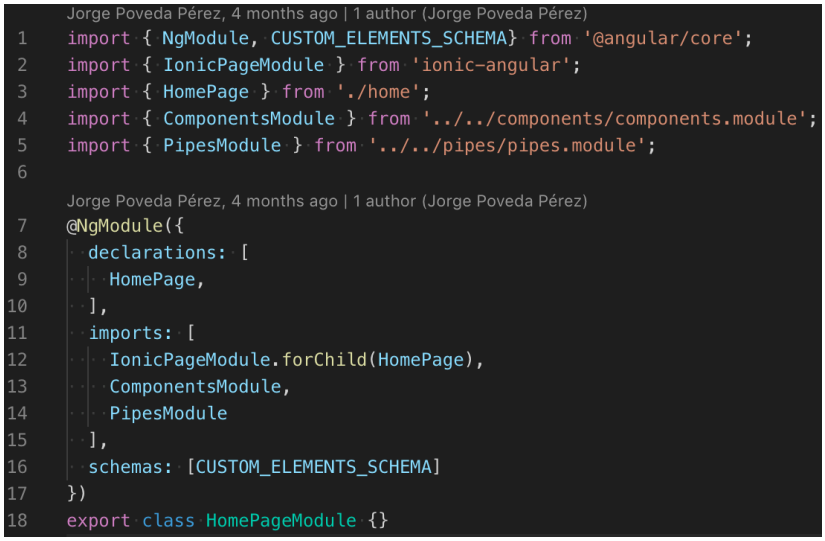

Ilustración 67: Captura del código de home.module.ts

- SCSS o Sassy Cascading Style Sheet: Encontramos los estilos específicos para esta página estructurados por clases que pueden ser citadas más tarde en nuestro HTML.

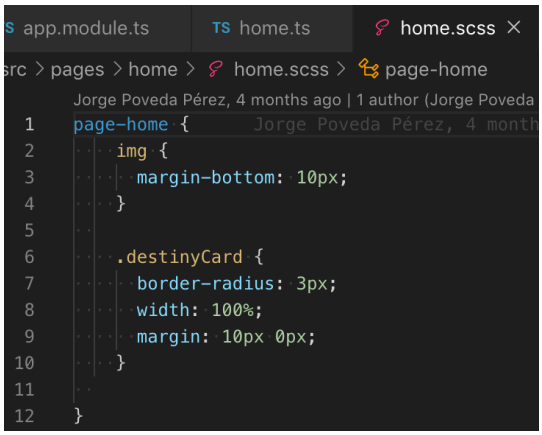

Ilustración 68: Captura del código de la página de estilos home.scss

Es importante destacar que existe una hoja de estilos global llamada app.scss en la que podemos englobar todos los estilos que no son solo específicos de una única página.

- TS: Aquí se programa la lógica, métodos y demás. A pesar de tratarse de una página, podemos observar que, siguiendo el patrón de Angular, se le aplica el decorador @Component(), que, a su vez, posee como metadatos el selector (nuestro archivo de estilos home.scss) y la plantilla (nuestro home.html).

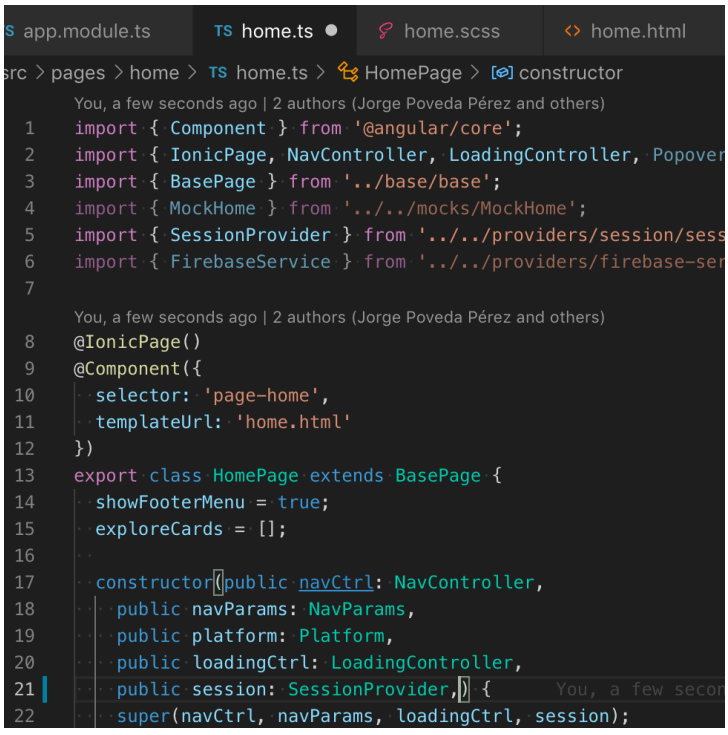

Ilustración 69: Captura del código de la página home.ts

Además, todas las páginas se extenderían a su vez de la página Base.

La página Base posee funcionalidades comunes a todas las páginas. Un claro ejemplo, es la sesión o el sistema de traducción e internacionalización de la App.

Para poseer una única instancia de Sesión y que ésta sea compartida en toda nuestra aplicación, nos aseguramos de instanciarla en el constructor de la página base.

Navegación

Lazy Load [27] es un sistema por el que las páginas de una aplicación solo se cargan cuando son requeridas.

La navegación entre las distintas páginas ha sido optimizada utilizando un sistema de "Lazy loading". Se ha decidido hacer así porque se cree que, para el caso de nuestra aplicación, es la opción más eficiente.

Sin esta característica, todas las páginas de una aplicación se ven instanciadas con su inicio. Esto provoca una sobrecarga de tiempo al cargar por primera vez la aplicación que para la experiencia de usuario puede resultar muy nociva.

El problema no se aprecia en aplicaciones de pequeño tamaño, pero se intensifica notablemente cuando pasamos de las 20 páginas o queremos aumentar los módulos y funcionalidades de nuestro programa.

Lazy Load no es siempre mejor, de hecho, si tuviéramos una aplicación con funcionalidades cerradas y no buscásemos escalabilidad o ampliaciones, no hacer uso de Lazy Load también sería una decisión decente. Al no hacerlo reduciríamos el tiempo de carga entre pantallas (de forma relativamente pequeña).

Internacionalización

Para las traducciones e internalización de la aplicación se hace uso de un elemento de Angular llamado Pipe. El comando de Ionic para crearlo seria:

$ ionic generate pipe translator

Un Pipe posee un método de transformación por el cual, dado un dato por parámetro, lo cambia por otro siguiendo la funcionalidad programada.

En el caso de la internacionalización de la aplicación encontramos que dado un diccionario por cada idioma (un objeto con formato JSON, por ejemplo), podríamos buscar la clave que llega por parámetro al método del Pipe y devolver el valor que tiene en el diccionario del idioma actual.

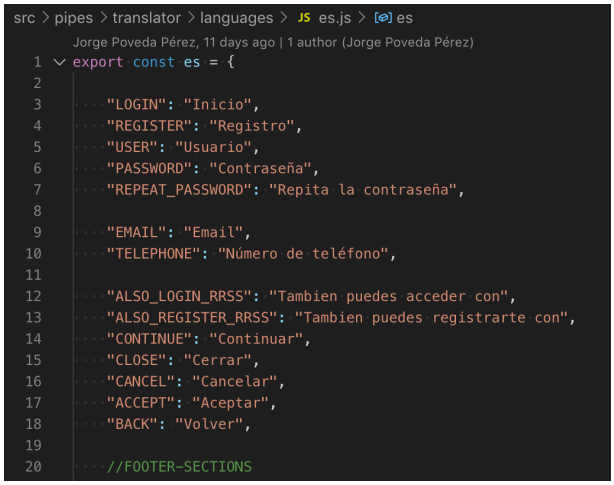

Ilustración 70: Captura de un trozo del diccionario "es"

A modo de ilustración, si al método transform le pasamos la clave ‘LOGIN’ y lenguaje ‘es’, veríamos como nos devuelve el string o cadena “Inicio”.

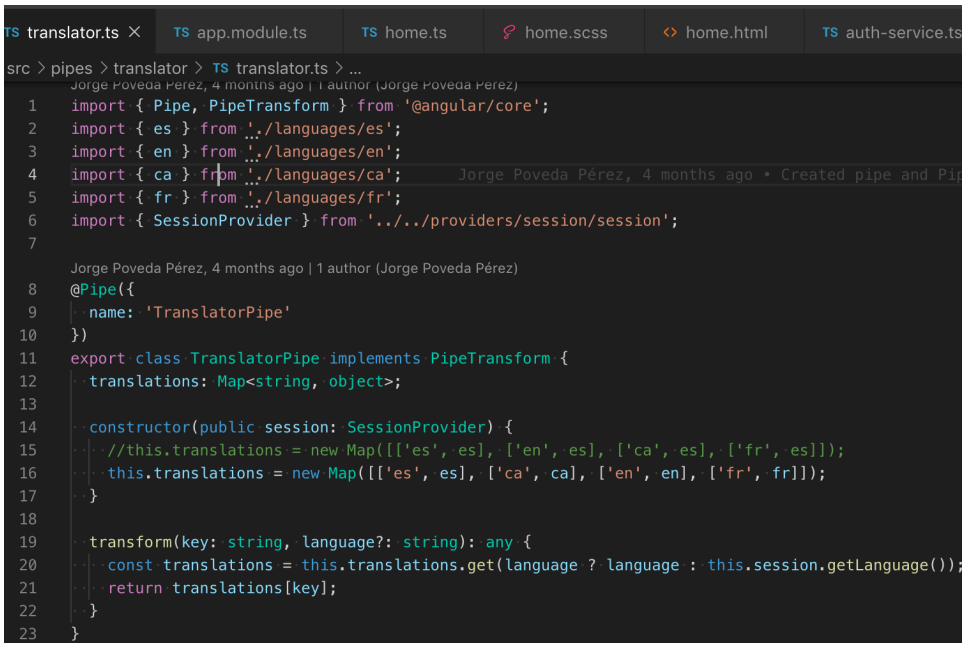

Ilustración 71: Captura del código TranslatorPipe.ts

Hasta aquí nos encontramos con un sistema de traducción bastante común y una funcionalidad que tampoco destaca por su elegancia.

No obstante, los Pipes en Angular poseen una característica y es que pueden ser llamados de forma muy sencilla también en sus plantillas HTML.

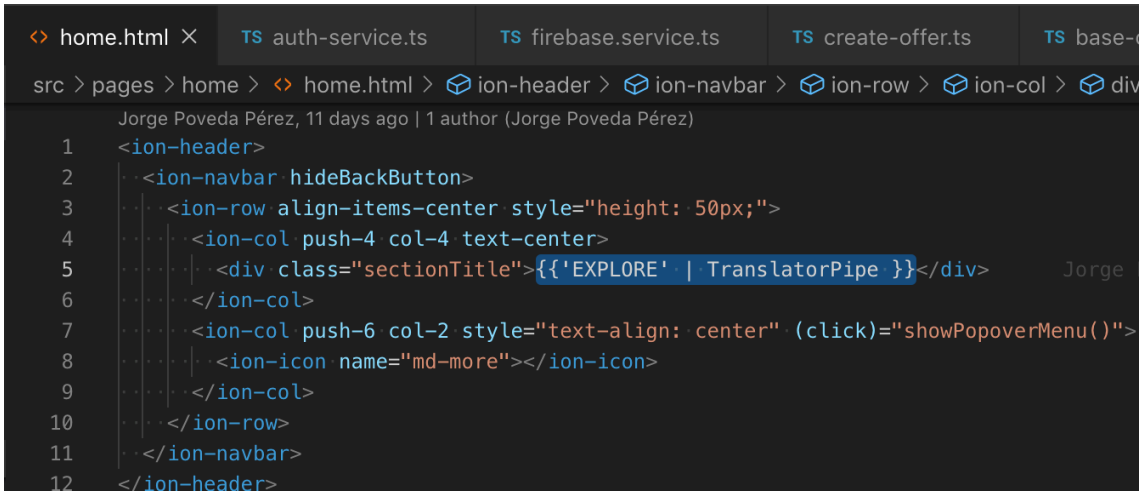

Ilustración 72: Captura mostrando un ejemplo de uso del Pipe en un .html

Con esa notación estaríamos obteniendo el valor de la clave EXPLORE en el diccionario del idioma que tiene la aplicación en sesión a la hora de renderizar la página HomePage.

Consumo de servicios

El consumo de los servicios de persistencia se lleva a cabo a través de Providers. Los providers pueden crearse con el comando:

$ ionic generate provider [name]

Estos providers son los que poseen la lógica de la Capa de Acceso a Datos. De forma usual, encontraríamos que los providers engloban llamadas HTTP a APIs y servicios externos.

En este caso es algo más particular, ya que al hacer uso directamente de Firestore, desaparece la necesidad de un API que trabaje de mediador entre las peticiones a BBDD y nuestra aplicación.

La transformación de la información, entonces, queda relegada de forma muy cómoda a Firebase Firestore quien nos devuelve en formato JSON la información que necesitamos para nuestra aplicación en Front.

Para entenderlo mejor, podemos analizar el flujo de datos en el momento de una autenticación correcta:

1. El usuario introduce las credenciales de forma correcta.
2. Recibe el id usuario de sus credenciales en Firestore y demás parámetros que establecen la seguridad de la transmisión.
3. Realiza una petición a Firestore para obtener los datos asociados con ese usuario en formato JSON y así configurar la sesión con el nombre, imagen de perfil, idioma predeterminado,…

Sistema de Roles

La aplicación, dados sus requerimientos necesita de un sistema de roles por el cual los usuarios sean categorizados en varios grupos.

Usuario normal

Se trata de un usuario registrado con la capacidad de buscar y reservar ofertas, así como enviar mensajes desde el chat tanto a soporte* como a otros usuarios, modificar los datos de su perfil, gestionar sus eventos, etc…

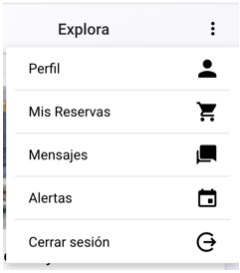

Ilustración 73: Captura del menú superior con un usuario normal

Usuario ofertador

Se trata de un usuario registrado que además de tener todas las funcionalidades de uno normal, es capaz de crear una oferta nueva.

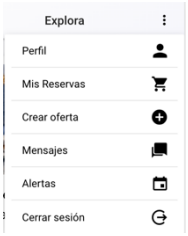

Ilustración 74: Captura del menú superior con un usuario ofertador

Usuario administrador

Se trata del administrador de la aplicación, es capaz de realizar todo lo anterior con el añadido de: contestar a los mensajes dirigidos a soporte*, dar de alta nuevos destinos y modificar el rol del resto de usuarios.

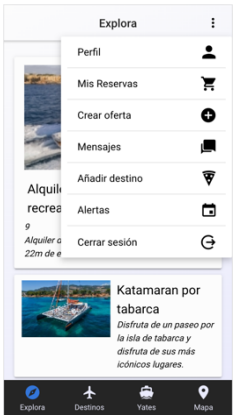

Ilustración 75: Captura del menú superior con un usuario administrador

Back-End

La parte de back-end de la aplicación está íntegramente implementada con Google Firebase.

Firebase nos garantiza cumplimentar varios requerimientos no funcionales de nuestra aplicación como la transmisión segura de los datos entre Back y Front-end, o la concurrencia simultánea de un gran número de usuarios.

Autenticación - FireAuth

Los usuarios se identifican en nuestra aplicación mediante un email y contraseña. Este proceso de validación lo realiza Firebase gracias a su módulo FireAuth.

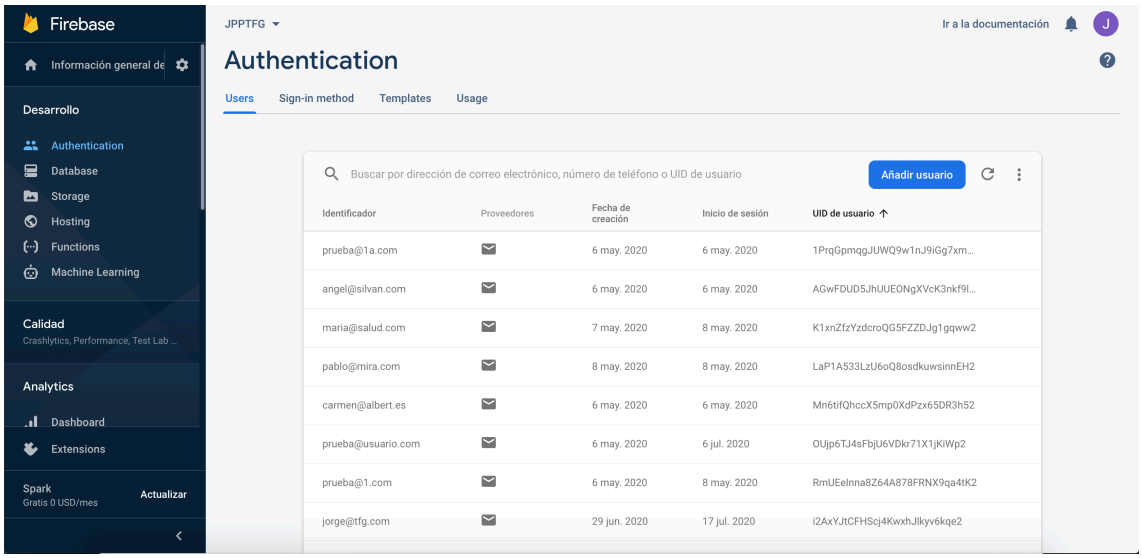

Ilustración 76: Captura del panel de administrador de Firebase en su sección Authentication

Una vez dado de alta el módulo en nuestra consola de desarrollador, añadiríamos los paquetes de Firebase a nuestro proyecto en Ionic e implementaríamos los métodos y lógica necesarios.

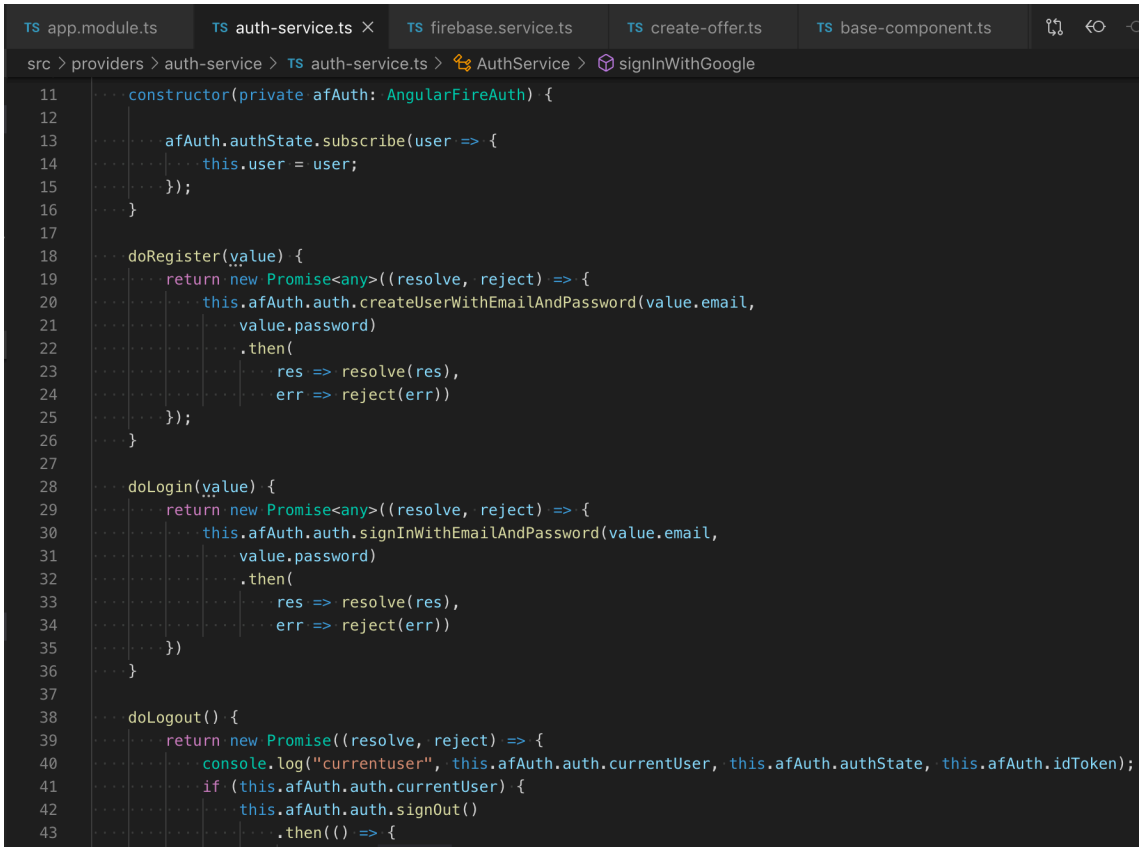

Ilustración 77: Captura del código requerido para comunicarse con Firebase Authentication

Para ello, crearíamos un Provider en nuestro proyecto que contendría la lógica de autenticación y servicios de credenciales en nuestra aplicación.

Ante una autenticación correcta, Firebase inicia una sesión para ese dispositivo y le asigna un token que solo es eliminado en el momento en el que el usuario acciona el botón de “Cerrar sesión” o el propio token expira.

La expiración del token se ha establecido en 36h siguiendo convencionalidades para este tipo de aplicaciones. Este tiempo viene determinado por la sensibilidad de los datos a los que podría acceder una persona maliciosa en el supuesto de que lograse obtener un token ajeno.

Pasado ese tiempo, tendríamos que reautenticarnos para poder seguir realizando peticiones.

El uso de una herramienta oficial de Google tan potente facilita la identificación con distintas redes sociales mediante un proceso de OAuth2.

BBDD - Firestore

Para la base de datos se ha utilizado Firestore. Se trata de una BBDD NoSQL que organiza los datos como “documentos”.

Para guardar información sobre un usuario, por ejemplo, la estructura sería la siguiente:

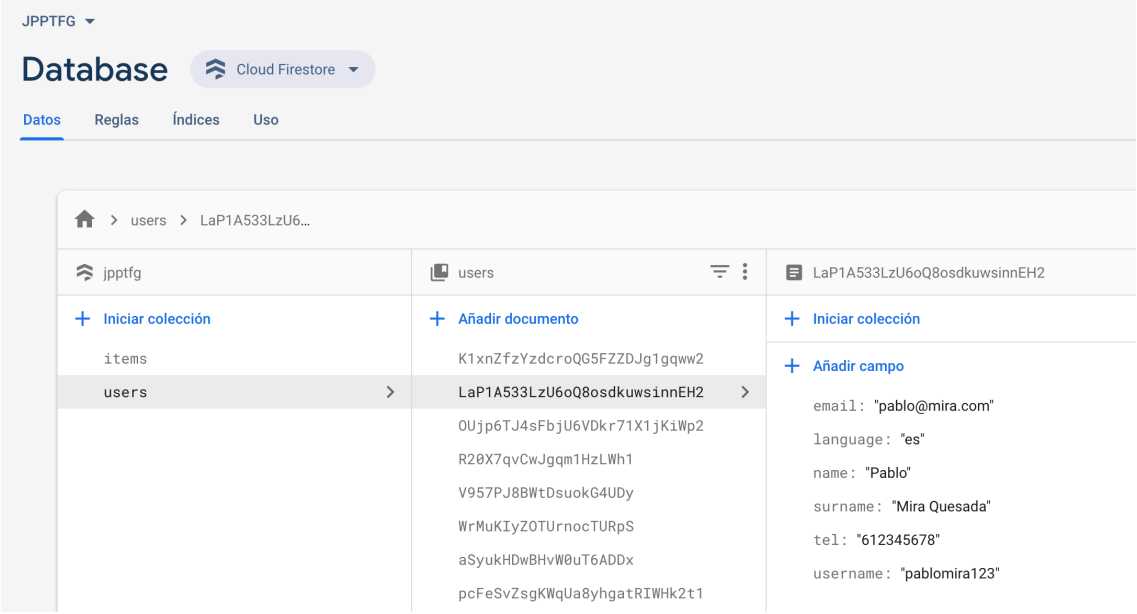

Ilustración 78: Captura de un ejemplo de BBDD con Firebase Firestore

Véase que los datos que podemos guardar en Firestore no incluyen credenciales de seguridad. Esto es así para garantizar la privacidad de los datos de inicio de sesión, que son guardados solo en la herramienta de Firebase Authentication.

Los servicios de esta Base de datos se consumen directamente desde nuestra App, la cual posee la capa de acceso a datos (CAD).

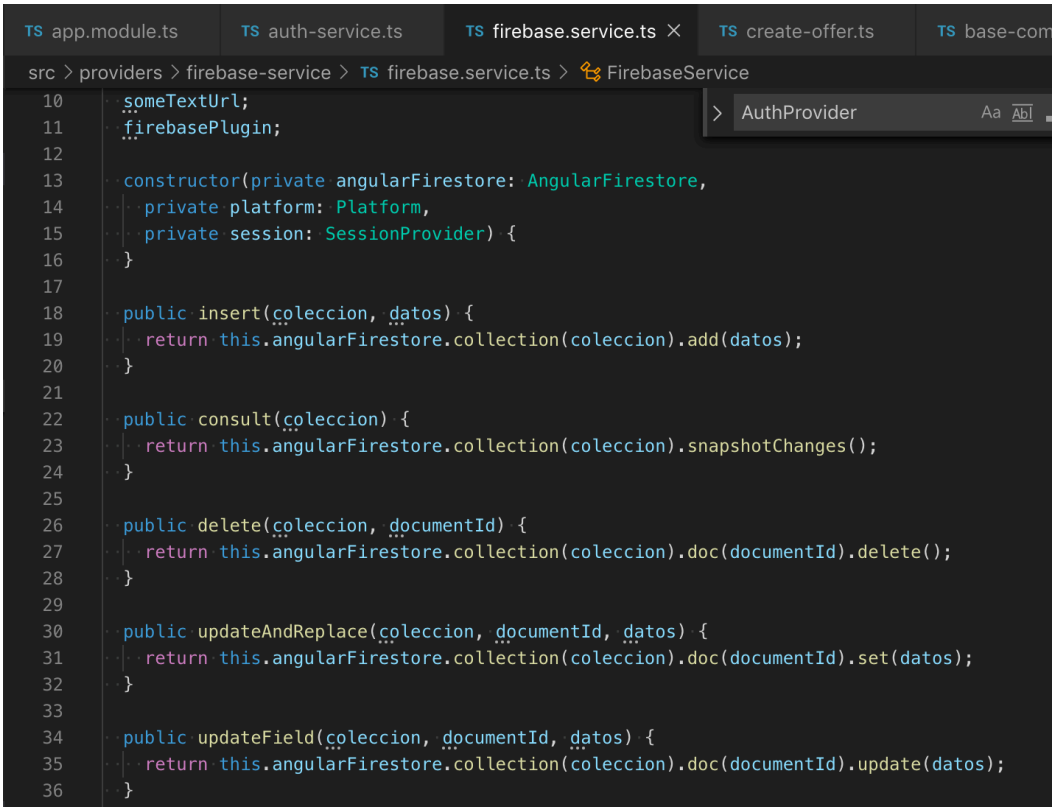

Ilustración 79: Captura del código necesario para hacer métodos CRUD

Para ilustrar un ejemplo, veríamos que en nuestra aplicación tenemos un Provider llamado firebase.service.ts que contiene los métodos básicos de toda BBDD, los métodos CRUD, en castellano; Crear, Leer, Actualizar y Borrar.

Gracias a los plugins y paquetes que Google pone a disposición de los desarrolladores, encontramos mucha facilidad a la hora de programar nuestra persistencia y lógica de datos, tanto es así, que podemos resumir sus métodos en una sola línea.

SEGURIDAD

Encontramos seguridad en la aplicación a varios niveles

En el acceso a los datos (BBDD)

La base de datos está implementada en Firebase [28], la plataforma de Google de desarrollo de aplicaciones.

Concretamente, nuestros datos son guardados en Firestore [29] que se trata de un contenedor de datos NoSQL que los divide por “documentos”.

Para acceder a los datos, Google aconseja y facilita el uso de reglas. Principalmente, se tratan de sentencias condicionales que ejecuta con cada request para verificar si la petición tiene acceso de lectura y/o escritura.

Al crear nuestra aplicación, Firebase nos proporciona una regla temporalmente. Pensada para el desarrollo y solo válida durante 30 días. Permite todas las peticiones entrantes tanto de lectura como de escritura.

Ilustración 80: Captura del panel de reglas de Firebase Firestore

Durante el desarrollo, y conforme se van añadiendo requisitos, conseguimos implementar Firebase Authentication [30].

Esta autenticación, y su estrecha vinculación con el resto de las herramientas de Firebase (en este caso, Firestore) nos da la opción de modificar y adaptar las reglas para aumentar la seguridad de las peticiones, de modo que la siguiente durante el desarrollo y de forma provisional, sería una que permita acceder y modificar todos los datos, pero solo si se trata petición que venga de un usuario identificado mediante Firebase Authentication.

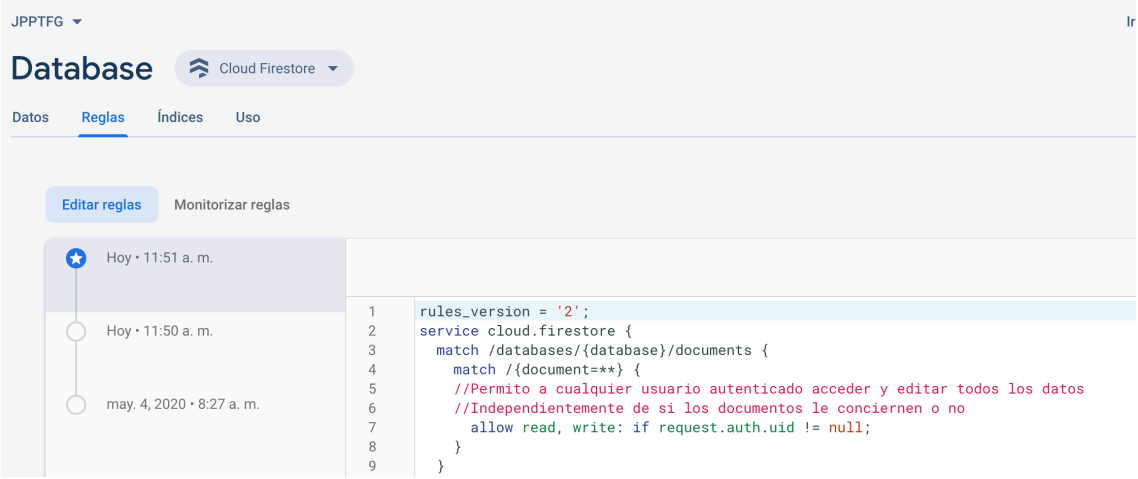

Ilustración 81: Captura del código de reglas para Firestore

El último paso sería codificar cotejar el "uid" de autenticación de la petición con el asociado a los documentos sensibles que se desean consumir.

En el dispositivo

En la app móvil también guardamos datos sensibles. Estos datos son almacenados en la memoria persistente del teléfono, y antes de hacerlo, son encriptados.

```
@Injectable()
export class DataStorageProvider {

  constructor(private nativeStorage: NativeStorage, private encryptor: EncryptorProvider) {
  }

  saveLogin(login) {
    let user = this.encryptor.encrypt(login.user)
    console.log("postEncript", user);

    this.nativeStorage.setItem(encryptKey.user, user)
      .then(
        () => console.log('Stored item!'),
        error => console.error('Error storing item', error)
      );
    let pass = this.encryptor.encrypt(login.password);
    this.nativeStorage.setItem(encryptKey.password, pass)
      .then(
        () => console.log('Stored item!'),
        error => console.error('Error storing item', error)
      );
  }
```

Ilustración 82: Captura del código de almacenamiento de datos en dispositivo

El algoritmo de encriptación utilizado es Advanced Encryption Standard (AES) [31], un algoritmo de criptografía simétrico estandarizado por EEUU y muy popular entre empresas e incluso gobiernos (la propia Estados Unidos, a través de la NSA ha aprobado su uso para el cifrado de información clasificada, eso sí, usando siempre claves superiores a los 192 bits).

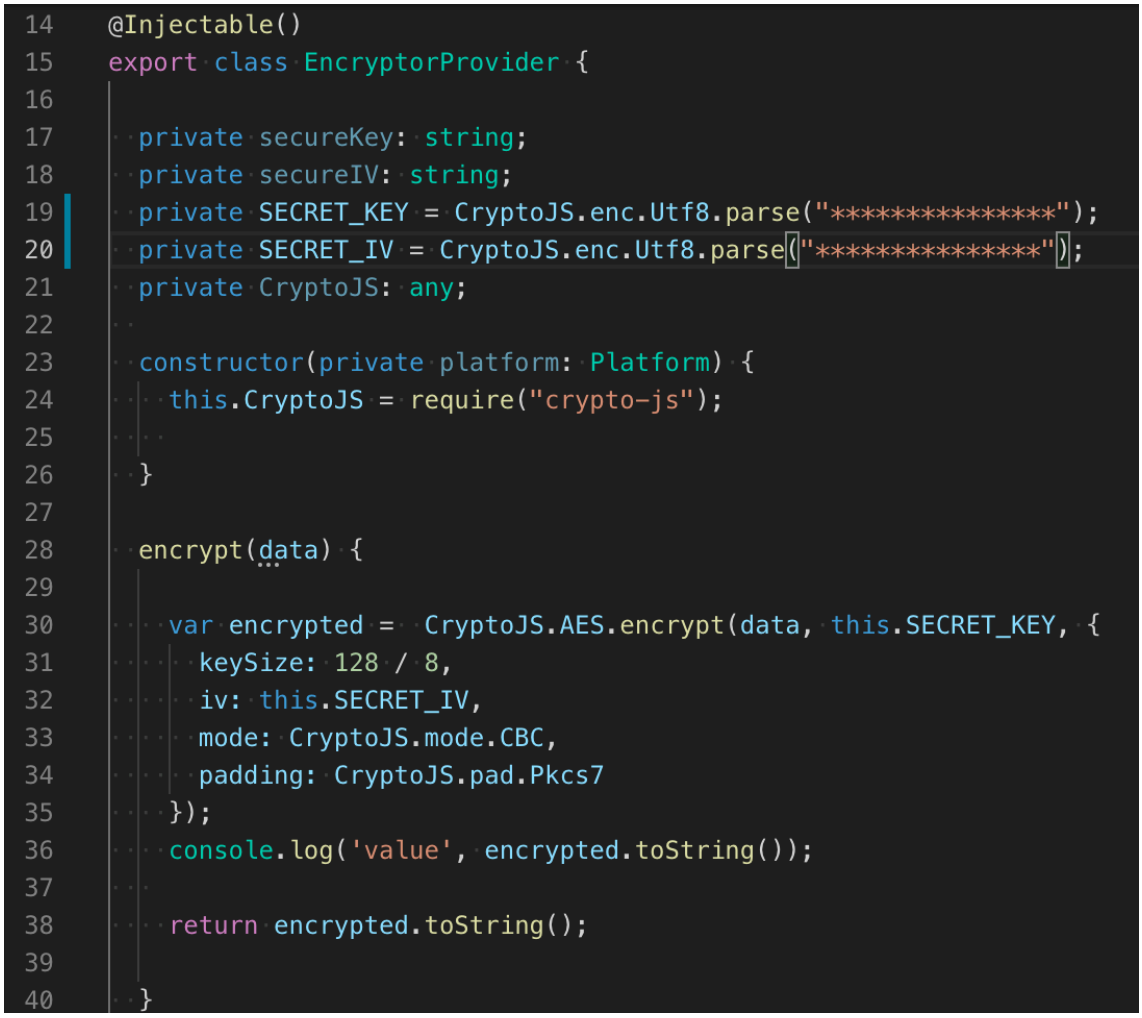

Ilustración 83: Captura del método de encriptación

El módulo crypto-js consta de varios métodos de encriptación, entre los cuales se ha elegido AES por su eficiencia, elegancia y el hecho de existen varias modalidades del mismo dependiendo del tamaño de la clave.

En la aplicación se ha hecho uso de AES-256 en detrimento de AES-128 y AES-192. Crypto-js nos permite elegir el tamaño dependiendo de la clave que le pasamos (en el caso del código sería SECRET_KEY).

APIS

Se ha hecho uso de varias APIs externas para la inclusión de ciertas funcionalidades.

Google Maps

Varios de los apartados de la aplicación poseen Mapas integrados. Google Maps ofrece varias formas de integrar su tecnología en dispositivos móviles.

Para aplicaciones nativas, encontraríamos un SDK descargable tanto para Android como iOS. No obstante, al tratarse de un software fundamentado en web lo haríamos inyectando una llamada a la API en la plantilla html del proyecto de la que se extienden todas nuestras vistas, el index.html.

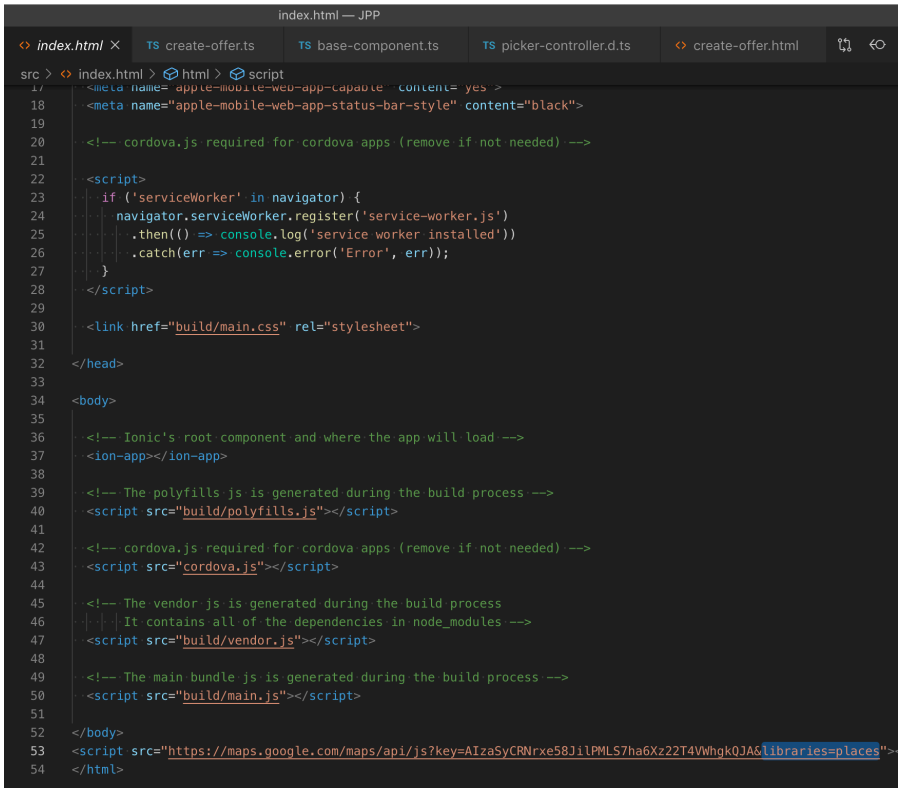

Ilustración 84: Captura del index.html, uno de los archivos raíz del proyecto Ionic

Este script podemos encontrarlo en nuestra consola de Firebase, más concretamente en su módulo de Maps y es necesario copiarlo y pegarlo en este fichero para hacer uso de esta API.

La “key” constituye el identificador de nuestro proyecto dado de alta en Firebase, mientras que el valor del parámetro libraries indica que además de la integración básica, queremos tener acceso a la librería Places de Google.

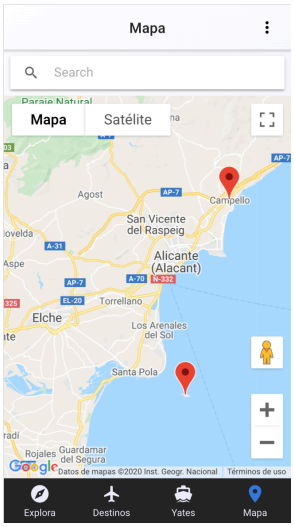

Ilustración 85: Pantalla Mapa con un dispositivo Android

En esta pantalla, encontramos un marcador que indica la localización de las ofertas.

La librería Places es importante ya que es la que nos ofrece los servicios de autocompletado, geolocalizaciones de lugares, ciudades, calles, etc,…

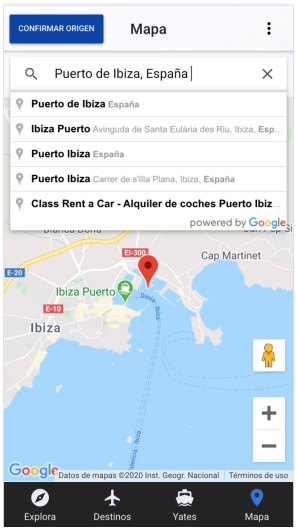

Ilustración 86: Pantalla Mapa con un dispositivo Android tras seleccionar un lugar de origen

Gracias a ella podemos hacer que el usuario añada marcadores de Maps a sus ofertas y así almacenar estos datos sobre el punto de origen y de destino, para más tarde, explotarlos aplicándoles filtros que nos ayuden a encontrar ofertas según la localización.

Facebook OAuth2

La aplicación integra autenticación con redes sociales. Para agregar Facebook, se da de alta una nueva aplicación en la consola de desarrolladores de Facebook.

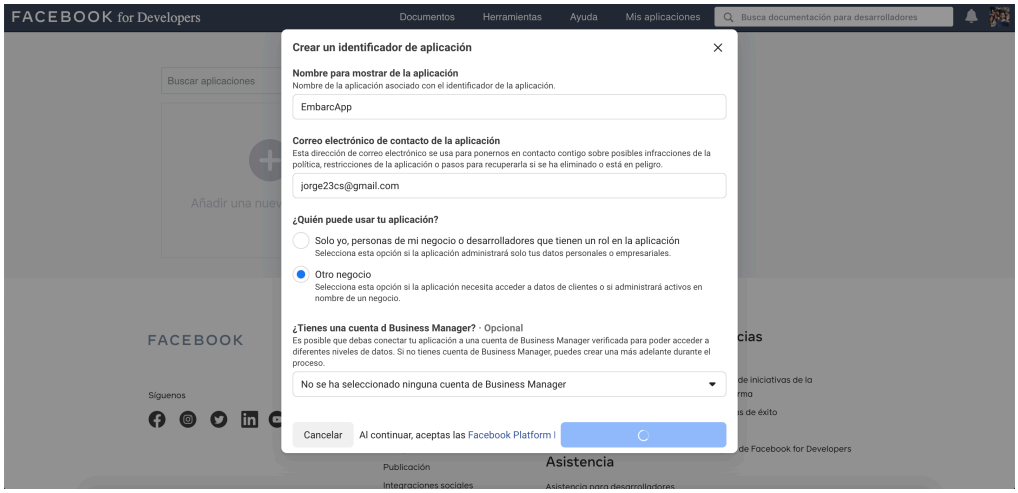

Ilustración 87: Creación de Aplicación en la consola de desarrolladores de Facebook

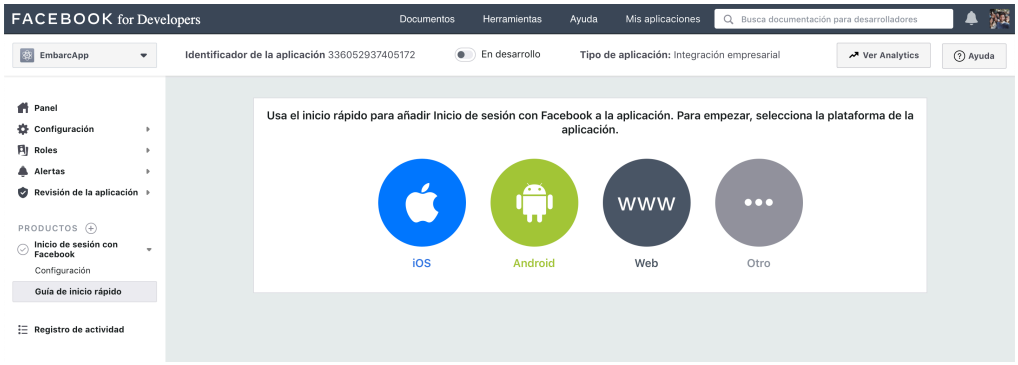

Ilustración 88: Captura del panel guía para añadir Inicio de sesión a una aplicación con Facebook

Y tras configurarla, tendríamos que añadir el servicio también en la consola de Firebase Authentication.

PRUEBAS

Durante el transcurso del desarrollo se han utilizado tres formas de validación.

Pruebas del aspecto según el dispositivo

La aplicación va dirigida a un amplio rango de dispositivos. El mismo código, la misma vista, se renderiza tanto en iOS como en Android como en el navegador (por medio de una PWA), por lo que es necesario validar que independientemente del dispositivo y su tamaño, el aspecto visual de la app sea el requerido.

Pruebas de funcionalidades

La aplicación está desarrollada en entorno web, por lo que si deseamos realizar pruebas y automatizarlas, necesitamos un IDE apropiado.

Se ha elegido Selenium [32] IDE. Se trata de una extensión de navegador que graba los eventos que suceden en la página y luego los reproduce.

Para generar los tests solo tendríamos que acceder al panel de la extensión, crear un nuevo proyecto y añadir Tests. Una vez creado, grabaremos las interacciones añadiendo “asserts” o comparaciones que demuestren que el funcionamiento es el especificado.

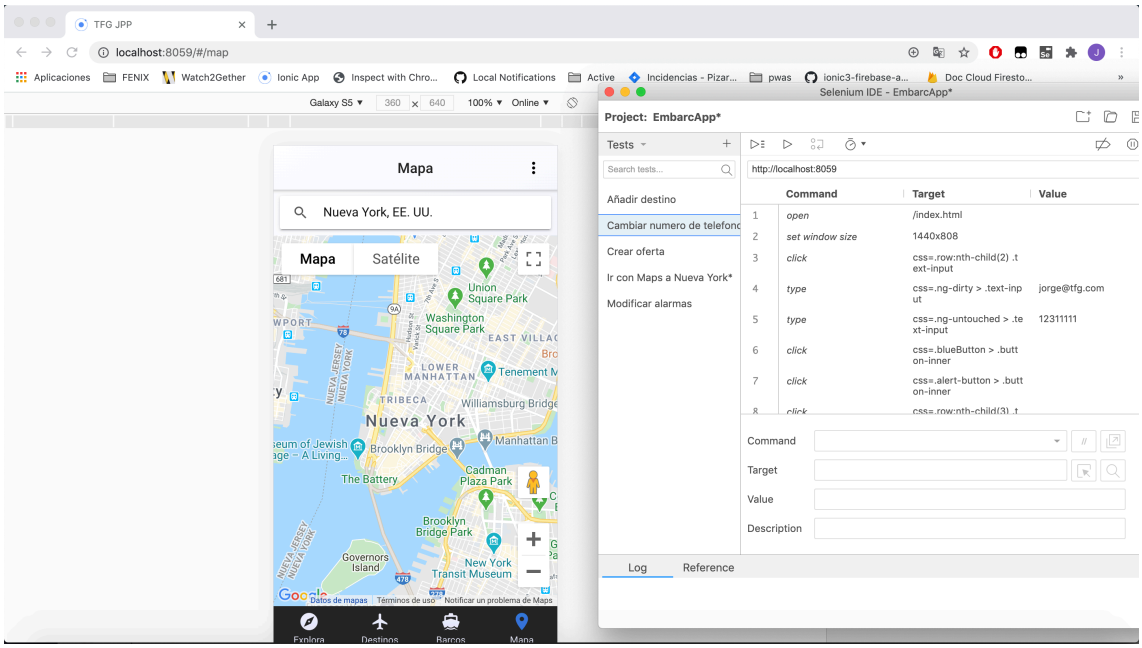

Ilustración 89: Captura de Selenium IDE con varias tareas grabadas.

Los tests pueden ejecutarse con solo pulsar un botón, por lo que es ideal para realizar y automatizar pruebas regresivas y de funcionalidad.

El uso de Selenium IDE favorece la creación de una aplicación más robusta y un entorno de pruebas de calidad.

Pruebas manuales

Al tratarse de una aplicación multiplataforma, en ocasiones encontramos funcionalidades con implementaciones muy dispares dependiendo de la plataforma. El ejemplo más claro sería el de los plugins de cordova.

Para probar funcionalidades como el acceso a la galería, la cámara o la posición del teléfono necesitamos hacer uso de propiedades del teléfono, por lo que, en muchas ocasiones, hasta que no probamos de forma estricta con un dispositivo físico, no somos capaces de validar con total seguridad la implementación.

CONCLUSIONES, TRABAJOS FUTUROS Y POSIBLES MEJORAS

Es imperativo tener en cuenta que en el proyecto no solo se ha propuesto como objetivo realizar un producto final atractivo, completo y viable, sino que también se ha considerado de vital importancia funcionar de forma pulcra, eficiente y eficaz durante el desarrollo.

Todos los medios utilizados en cada etapa han sido previamente estudiados, considerados y seleccionados para conseguir no solo un buen producto, si no aplicar y afianzar unas buenas prácticas, metodologías y aptitudes a la hora de trabajar.

En el momento en el que se presenta la descripción del proyecto se detectan varios requisitos fundamentales. El primero y más importante es el de implementar una solución informática disponible en navegador (PWA), y móvil (iOS y Android).

Un camino convencional, hubiese supuesto un imposible al considerar tan solo un desarrollador para tres soluciones totalmente independientes en Front-End.

Al principio se consideró la posibilidad de dejar de lado una de las plataformas móviles, pero dada la importancia que se le da a la accesibilidad de la aplicación, se decidió buscar otras alternativas.

Finalmente se encontró Ionic, un framework que ha demostrado madurez y capacidad para presentar un producto final que no tiene nada que envidiar al que podemos ver en un desarrollo nativo.

Esta elección ha permitido que la aplicación pueda desarrollarse de forma más creativa al tener mucho más tiempo y libertad para poder concretar y añadir funcionalidades que en un principio no estaban contempladas.

Huelga decir que, aunque la creatividad es algo indispensable durante la producción de un software, también puede ser un arma de doble filo. En ocasiones, vemos proyectos que divagan de forma desacertada al seguir un camino poco marcado y difuso, donde se intenta “morder más de lo que se puede masticar” realizando tareas frívolas en detrimento de las que realmente son prioritarias.

Para evitar que algo así pudiera suceder se ha utilizado un sistema fundamentado en las metodologías ágiles donde los requisitos se priorizaban y estimaban sin olvidar nunca el valor real que aportaban al producto.

El resultado final de todo este proceso ha sido un producto maduro, atractivo, escalable, que podría ser comercializado.

Posibles mejoras

Las posibilidades de mejora cubren un amplio rango de funcionalidades:

- Incorporación de un módulo de pagos en línea con un sistema de TPV al integrar una API externa que permita realizar ingresos con tarjeta.
- Creación de una Inteligencia Artificial que sea capaz de consultar e interpretar datos del historial del usuario en la app o su geolocalización para mostrarle los resultados que más le puedan interesar.
- Durante el desarrollo se contempló la posibilidad de que la aplicación obtuviese datos de ofertas de APIs externas. Si se dispusiese del capital requerido para poder consumir los servicios de alguna de estas APIs, podríamos obtener y mostrar un listado de ofertas considerablemente mayor.

Monetización

Se han estudiado varias posibilidades para conseguir remuneración económica a través de la App.

Actualmente, la más realista consistiría en cobrar una pequeña suma de dinero a los usuarios que crean ofertas en nuestra aplicación.

Mientras no exista un módulo de pagos integrado en la aplicación no es plausible cobrar una comisión por reserva, por lo que, la mejor forma de conseguir ingresos sería a través de publicidad de terceros, tarifas en la creación de ofertas e incluso promoción de dichas ofertas para conseguir que aparezcan en los apartados como las más destacadas.

INDICE DE FIGURAS

REFERENCIAS

[1] Chárter Naútico: https://es.wikipedia.org/wiki/Ch%C3%A1rter_n%C3%A1utico
[2] Turismo acuático: https://www.opinionociohoteles.com/que-turismo-acuatico
[3] Repositorio que contiene el proyecto: https://bitbucket.org/jpptfg/embarcapp
[4] User Experience: https://es.wikipedia.org/wiki/Experiencia_de_usuario
[5] Móviles según SO: https://gs.statcounter.com/os-market-share/mobile/worldwide
[6] PWA: https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web_progresiva
[7] Informe de la cámara de comercio balear (pág. 20): https://www.cambramallorca.com/documentos/Desp_1045.pdf
[8] Volumen de negocio según la CAIB: http://www.caib.es/pidip2front/jsp/es/ficha-convocatoria/las-272-empresas-naacuteuticas-de-las-islas-baleares-generan-un-volumen-de-negocio-de-561-millones-de-euros
[9] Nautal: https://www.nautal.es/
[10] Nautal Owners iOS: https://apps.apple.com/es/app/nautal-owners/id1162101748?l=en
[11] Nautal Owners Android: https://play.google.com/store/apps/details?id=com.nautalapp&hl=es
[12] Y4ybooking: https://www.y4ybooking.com/en
[13] Nausys: https://www.nausys.com/
[14] LeafletJS: https://leafletjs.com/
[15] OpenStreetMap: https://www.openstreetmap.org/
[16] Logitravel: https://www.logitravel.com/
[17] Apache Cordova: https://cordova.apache.org/docs/es/latest/guide/overview/
[18] Ionic: https://ionicframework.com/
[19] Angular: https://angular.io/guide/architecture
[20] CLI: https://www.w3schools.com/whatis/whatis_cli.asp
[21] Gradle: https://es.wikipedia.org/wiki/Gradle
[22] NPM: https://docs.npmjs.com/about-npm/
[23] ¿Qué es BitBucket?: https://www.atlassian.com/es/software/bitbucket
[24] Git: https://es.wikipedia.org/wiki/Git
[25] Trello: https://es.wikipedia.org/wiki/Trello
[26] Kanban, ¿qué es y como aplicarlo al desarrollo software?: https://www.viewnext.com/kanban-desarrollo-software/
[27] Lazy Load: https://ionicframework.com/blog/ionic-and-lazy-loading-pt-1/
[28] Google Firebase: https://es.wikipedia.org/wiki/Firebase
[29] Google's Realtime DB vs Firestore: https://firebase.google.com/docs/database/rtdb-vs-firestore?hl=es-419
[30] Firebase Auth: https://firebase.google.com/docs/auth?hl=es
[31] AES: https://es.wikipedia.org/wiki/Advanced_Encryption_Standard
[32] Selenium: https://es.wikipedia.org/wiki/Selenium